L Number	Hits	Search Text	DB	Time stamp
1	7014559	number or value or amount	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 13:24
2	9926667	count$4 or sum$4 or list$4 or determin$4 or group$4 or total$4 or includ$4 or track$4 or check$4	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 13:25
4	2882045	active or running or operational or execut$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 13:26
6	9543594	process$5 or task$4 or program$ or executabl$ or operation$ or cod$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 13:27
7	7716414	start$5 or stop$5 or enter$5 or exit$5 or begin$5 or end$5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 13:27
8	2353449	(number or value or amount) with (count$4 or sum$4 or list$4 or determin$4 or group$4 or total$4 or includ$4 or track$4 or check$4)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 13:30
9	1127756	(active or running or operational or execut$5) with (process$5 or task$4 or program$ or executabl$ or operation$ or cod$5)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 13:32
10	59707	((number or value or amount) with (count$4 or sum$4 or list$4 or determin$4 or group$4 or total$4 or includ$4 or track$4 or check$4)) with ((active or running or operational or execut$5) with (process$5 or task$4 or program$ or executabl$ or operation$ or cod$5))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 13:32
11	13694	(((number or value or amount) with (count$4 or sum$4 or list$4 or determin$4 or group$4 or total$4 or includ$4 or track$4 or check$4)) with ((active or running or operational or execut$5) with (process$5 or task$4 or program$ or executabl$ or operation$ or cod$5))) same (start$5 or stop$5 or enter$5 or exit$5 or begin$5 or end$5)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 13:33
12	26161	709/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 13:34
13	19553	711/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 13:35
14	17819	710/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 13:35
15	6289	703/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 13:35

16	63909	709/$.ccls. or 711/$.ccls. or 710/$.ccls. or 703/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 13:36
17	1179	(709/$.ccls. or 711/$.ccls. or 710/$.ccls. or 703/$.ccls.) and ((((number or value or amount) with (count$4 or sum$4 or list$4 or determin$4 or group$4 or total$4 or includ$4 or track$4 or check$4)) with ((active or running or operational or execut$5) with (process$5 or task$4 or program$ or executabl$ or operation$ or cod$5))) same (start$5 or stop$5 or enter$5 or exit$5 or begin$5 or end$5))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 13:37
18	661	((709/$.ccls. or 711/$.ccls. or 710/$.ccls. or 703/$.ccls.) and ((((number or value or amount) with (count$4 or sum$4 or list$4 or determin$4 or group$4 or total$4 or includ$4 or track$4 or check$4)) with ((active or running or operational or execut$5) with (process$5 or task$4 or program$ or executabl$ or operation$ or cod$5))) same (start$5 or stop$5 or enter$5 or exit$5 or begin$5 or end$5))) and count	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 13:37
19	544	(((709/$.ccls. or 711/$.ccls. or 710/$.ccls. or 703/$.ccls.) and ((((number or value or amount) with (count$4 or sum$4 or list$4 or determin$4 or group$4 or total$4 or includ$4 or track$4 or check$4)) with ((active or running or operational or execut$5) with (process$5 or task$4 or program$ or executabl$ or operation$ or cod$5))) same (start$5 or stop$5 or enter$5 or exit$5 or begin$5 or end$5))) and count) and processor	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 13:37
20	446	((((709/$.ccls. or 711/$.ccls. or 710/$.ccls. or 703/$.ccls.) and ((((number or value or amount) with (count$4 or sum$4 or list$4 or determin$4 or group$4 or total$4 or includ$4 or track$4 or check$4)) with ((active or running or operational or execut$5) with (process$5 or task$4 or program$ or executabl$ or operation$ or cod$5))) same (start$5 or stop$5 or enter$5 or exit$5 or begin$5 or end$5))) and count) and processor) and @ad<20000331	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 14:10
21	122	(((((709/$.ccls. or 711/$.ccls. or 710/$.ccls. or 703/$.ccls.) and ((((number or value or amount) with (count$4 or sum$4 or list$4 or determin$4 or group$4 or total$4 or includ$4 or track$4 or check$4)) with ((active or running or operational or execut$5) with (process$5 or task$4 or program$ or executabl$ or operation$ or cod$5))) same (start$5 or stop$5 or enter$5 or exit$5 or begin$5 or end$5))) and count) and processor) and @ad<20000331) and (count with processor)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 13:39

22	7	((((((709/$.ccls. or 711/$.ccls. or 710/$.ccls. or 703/$.ccls.) and ((((number or value or amount) with (count$4 or sum$4 or list$4 or determin$4 or group$4 or total$4 or includ$4 or track$4 or check$4)) with ((active or running or operational or execut$5) with (process$5 or task$4 or program$ or executabl$ or operation$ or cod$5))) same (start$5 or stop$5 or enter$5 or exit$5 or begin$5 or end$5))) and count) and processor) and @ad<20000331) and (count with processor)) and (count with exit)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 13:58
23	8	((((((709/$.ccls. or 711/$.ccls. or 710/$.ccls. or 703/$.ccls.) and ((((number or value or amount) with (count$4 or sum$4 or list$4 or determin$4 or group$4 or total$4 or includ$4 or track$4 or check$4)) with ((active or running or operational or execut$5) with (process$5 or task$4 or program$ or executabl$ or operation$ or cod$5))) same (start$5 or stop$5 or enter$5 or exit$5 or begin$5 or end$5))) and count) and processor) and @ad<20000331) and (count with processor)) and "logical processor"	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 14:11
24	899	((709/$.ccls. or 711/$.ccls. or 710/$.ccls. or 703/$.ccls.) and ((((number or value or amount) with (count$4 or sum$4 or list$4 or determin$4 or group$4 or total$4 or includ$4 or track$4 or check$4)) with ((active or running or operational or execut$5) with (process$5 or task$4 or program$ or executabl$ or operation$ or cod$5))) same (start$5 or stop$5 or enter$5 or exit$5 or begin$5 or end$5))) and @ad<20000331	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 14:10
25	19	(((709/$.ccls. or 711/$.ccls. or 710/$.ccls. or 703/$.ccls.) and ((((number or value or amount) with (count$4 or sum$4 or list$4 or determin$4 or group$4 or total$4 or includ$4 or track$4 or check$4)) with ((active or running or operational or execut$5) with (process$5 or task$4 or program$ or executabl$ or operation$ or cod$5))) same (start$5 or stop$5 or enter$5 or exit$5 or begin$5 or end$5))) and @ad<20000331) and "logical processor"	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/01/28 14:11

US006272533B1

(12) **United States Patent**
Browne

(10) **Patent No.: US 6,272,533 B1**
(45) **Date of Patent: Aug. 7, 2001**

(54) **SECURE COMPUTER SYSTEM AND METHOD OF PROVIDING SECURE ACCESS TO A COMPUTER SYSTEM INCLUDING A STAND ALONE SWITCH OPERABLE TO INHIBIT DATA CORRUPTION ON A STORAGE DEVICE**

(76) Inventor: **Hendrik A. Browne**, 6211 Florence Way, Alexandria, VA (US) 22310

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/250,277**

(22) Filed: **Feb. 16, 1999**

(51) **Int. Cl.**[7] **G06F 13/38**
(52) **U.S. Cl.** **709/213**; 710/128; 711/152; 711/100; 711/111; 711/154
(58) **Field of Search** 711/100, 111, 711/112, 154, 152; 360/60; 710/128, 131; 709/213

(56) **References Cited**

U.S. PATENT DOCUMENTS

Re. 33,328	*	9/1990	Director	360/60 X
4,750,111	*	6/1988	Crosby, Jr. et al.	710/58
5,166,939	*	11/1992	Jaffe et al.	714/766
5,268,960	*	12/1993	Hung et al.	380/4
5,765,034	*	6/1998	Recio	710/131 X
5,920,893	*	7/1999	Nakayama et al.	709/213 X
5,991,829	*	11/1999	Giorgio et al.	709/213 X

FOREIGN PATENT DOCUMENTS

93/02419 * 2/1993 (WO) .

OTHER PUBLICATIONS

Control Data Corporation CDC Fixed Storage, vol. 2, Hardware Maintenance Manual, pp. i, ii, 1–1, 1–2, 1–22 through 1–30 and 1–160 through 1–165; 1984.*

Micron Electronics, Inc., Micron System User's Guide for Millennia and ClientPro Systems, 1998, pp. 4–19.

The RAID Advisory Board, Inc., The RAID Book, 6th ed., 1997, p. 16.

Janah, Monua, "The Cost of Networking," Information Week, Oct. 19, 1998, pp. 48 et seq.

Penenberg, Adam L., "We were long gone when he pulled the plug," Forbes, Nov. 16, 1998, pp. 134 et seq. (More legible copy of article taken from Forbes web site also attached.).

The PC Technology Guide, Motherboards, updated Oct. 12, 1998, http://www.dircon.co.uk/pctechguide.

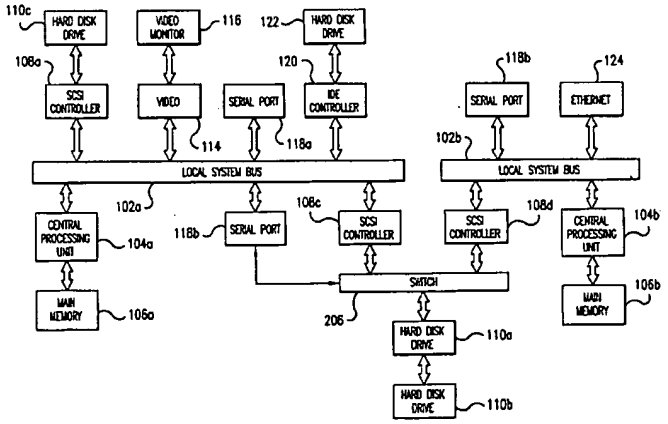
(List continued on next page.)

Primary Examiner—David L. Robertson
(74) *Attorney, Agent, or Firm*—Fulbright & Jaworski LLP

(57) **ABSTRACT**

A computer system includes hardware for selectively disabling alteration of data residing on a mass storage device which is subject to remote access. In one embodiment, a hard disk drive is operated in a conventional manner including both read and write modes when the system is being operated in a non-secure mode of operation, such as when remote access is not allowed. In a secure mode of operation, a locally operated switch is used to disable writing to the hard disk drive to maintain data integrity on the drive. The system may also include first and second electrically isolated buses and corresponding processors. In this configuration, the hard disk drive may be selectively connected to the first bus and processor for the storage of data, or to the second bus and processor when in a secure mode to provide for read-only remote access to the information stored on the hard drive. A write-only hard drive may also be included for storage of confidential information provided by remote users so that other remote users cannot access that information. In a master/slave processor configuration, all system programming is resident in an isolated portion of the system inaccessible to remote users. The slave processor receives instructions only from the master processor so that the operation of the slave processor cannot be compromised by viruses uploaded by remote users.

45 Claims, 9 Drawing Sheets



OTHER PUBLICATIONS

Intel Motherboard General FAQ, updated Aug. 31, 1998, http://www.connectedpc.com. (*Legal Information* © 1998 Intel Corp.).

The PC Technology Guide, Processors, updated Oct. 6, 1998, http://www.dircon.co.uk/pctechguide.

PICMG®—PCI–ISA, 1998, http://www.picmg.com (© 1998 PICMG).

PICMG ISA/PCI Passive Backplane (source information unavailable).

The PC Guide, System Bus Functions and Features, Site Version 1.7.2—Version Date: Sep. 20, 1998, http://www-.pcguide.com (© Copyright 1997–98, Charles M. Kozierok).

The PC Guide, Older Bus Types, Site Version 1.7.2—Version Date: Sep. 20, 1998,http://www.pcguide.com (© Copyright 1997–98, Charles M. Kozierok).

The PC Guide, Peripheral Component Interconnect (PCI) Local Bus, Site Version 1.7.2—Version Date: Sep. 20, 1998, http://www.pcguide.com (© Copyright 1997–98, Charles M. Kozierok).

The PC Guide, System Buses, Site Version 1.7.2—Version Date: Sep. 20, 1998,http://www.pcguide.com (© Copyright 1997–98, Charles M. Kozierok).

PC Webopaedia, bus, Last modified May 14, 1998,http://www.webopedia.internet.com (© 1998 Mecklermedia Corporation.

PC Webopaedia, Industry Standard Architecture (ISA) bus, Last modified May 15, 1998, http://www.webopedia.internet.com (© 1998 Mecklermedia Corporation).

PC Webopaedia, PCI, Last modified May 19, 1998,http://www.webopedia.internet.com (© 1998 Mecklermedia Corporation).

PC Webopaedia, local bus, Last modified May 19, 1998, http://www.webopedia.internet.com (© 1998 Mecklermedia Corporation).

PC Webopaedia, expansion bus, Last modified Dec. 5, 1998,http://www.webopedia.internet.com (© 1998 Mecklermedia Corporation).

PC Webopaedia, external bus, Last modified Dec. 30, 1997, http://www.webopedia.internet.com (© 1998 Mecklermedia Corporation).

PC Webopaedia, SCSI, Last modified Oct. 1, 1998,http://www.webopedia.internet.com (© 1998 Mecklermedia Corporation.

The PC Guide, Hard Disk General Interface Factors, Site Version 1.7.2—Version Date: Sep. 20, 1998, http://www-.pcguide.com (© Copyright 1997–98, Charles M. Kozierok).

Ögren, Joakim and Williams, Dan, Connector, IDE Internal Connector, http://www.margo.student.utwente.nl/stefan/hwb/co_IdeInternal.

* cited by examiner-

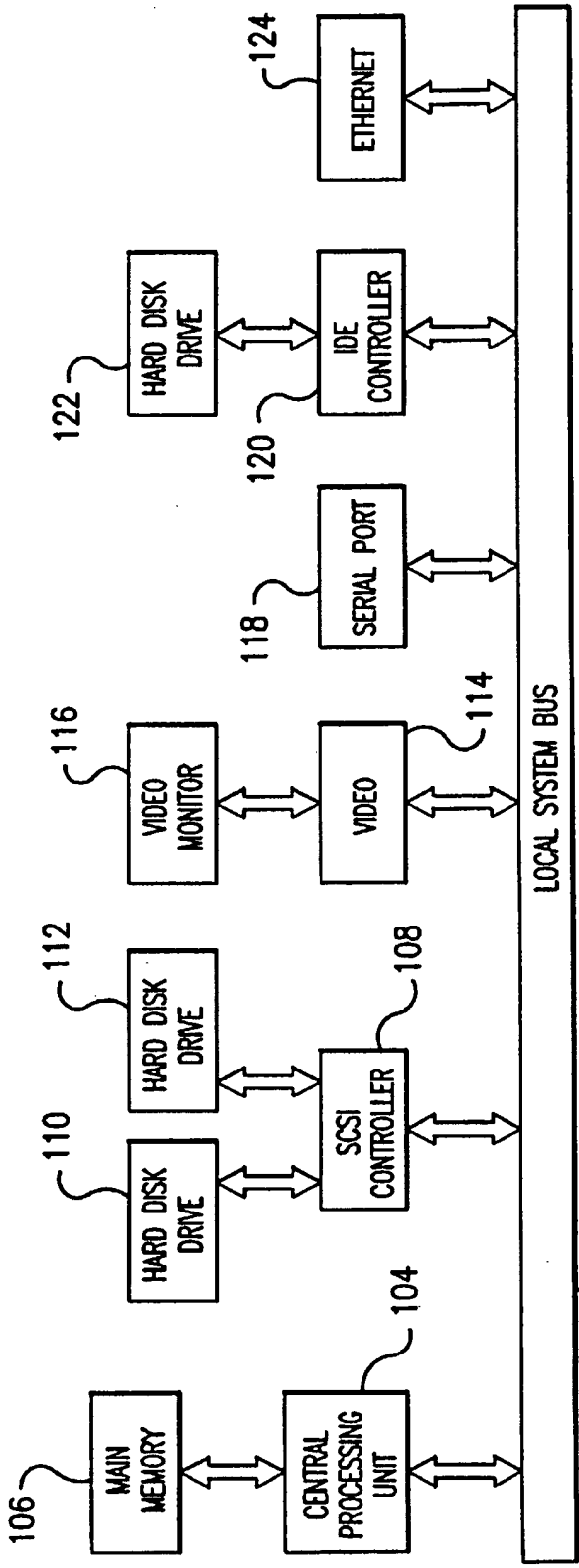


FIG. 1
PRIOR ART

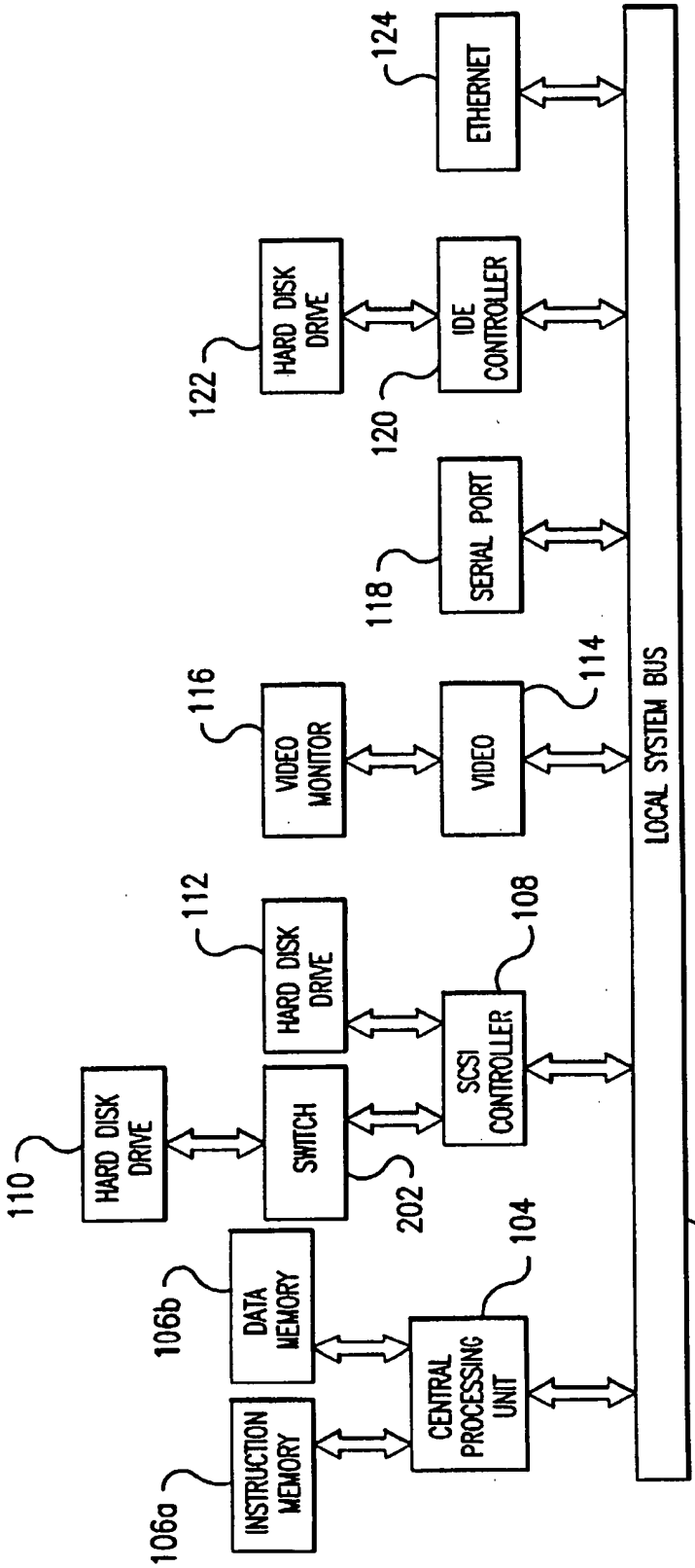
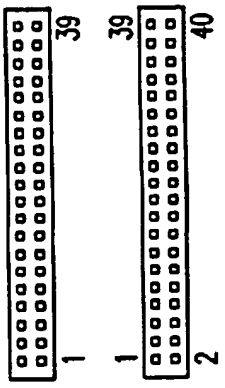

ETHERNET
124
HARD DISK DRIVE
122
IDE CONTROLLER
120
SERIAL PORT
118
VIDEO MONITOR
116
VIDEO
114
LOCAL SYSTEM BUS
HARD DISK DRIVE
112
SCSI CONTROLLER
108
HARD DISK DRIVE
110
SWITCH
202
DATA MEMORY
106b
CENTRAL PROCESSING UNIT
104
INSTRUCTION MEMORY
106a
102
200

FIG.2



PIN	NAME	DIR.	DESCRIPTION
1	/RESET		RESET
2	GND	⇧	GROUND
3	DD7	⇕	DATA 7
4	DD8	⇕	DATA 8
5	DD6	⇕	DATA 6
6	DD9	⇕	DATA 9
7	DD5	⇕	DATA 5
8	DD10	⇕	DATA 10
9	DD4	⇕	DATA 4
10	DD11	⇕	DATA 11
11	DD3	⇕	DATA 3
12	DD12	⇕	DATA 12
13	DD2	⇕	DATA 2
14	DD13	⇕	DATA 13
15	DD1	⇕	DATA 1

PIN	NAME	DIR.	DESCRIPTION
16	DD14	⇕	DATA 14
17	DD0	⇕	DATA 0
18	DD15	⇕	DATA 15
19	GND	⇕	GROUND
20	KEY		KEY
21	N/C		NOT CONNECTED
22	GND		GROUND
23	/IOW	⇧	READ STROBE
24	GND		GROUND
25	/IOR	⇧	READ STROBE
26	GND		GROUND
27	IO_RDY	⇩	I/O CHANNEL
28	ALE	⇧	ADDRESS LATCH
29	N/C		NOT CONNECTED
30	GND		GROUND

PIN	NAME	DIR.	DESCRIPTION
31	IRQR	⇩	INTERRUPT REQUEST
32	/IOCS16		I/O CHIP SELECT
33	DA1		ADDRESS
34	N/C		NOT CONNECTED
35	DA0	⇧	ADDRESS 0
36	DA2	⇧	ADDRESS 2
37	/IDE_CS0	⇧	(1F0-1F7)
38	/IDE_CS1	⇧	(3F6-3F1)
39	/ACTIVE	⇧	LED DRIVER
40	GND		GROUND

FIG.3

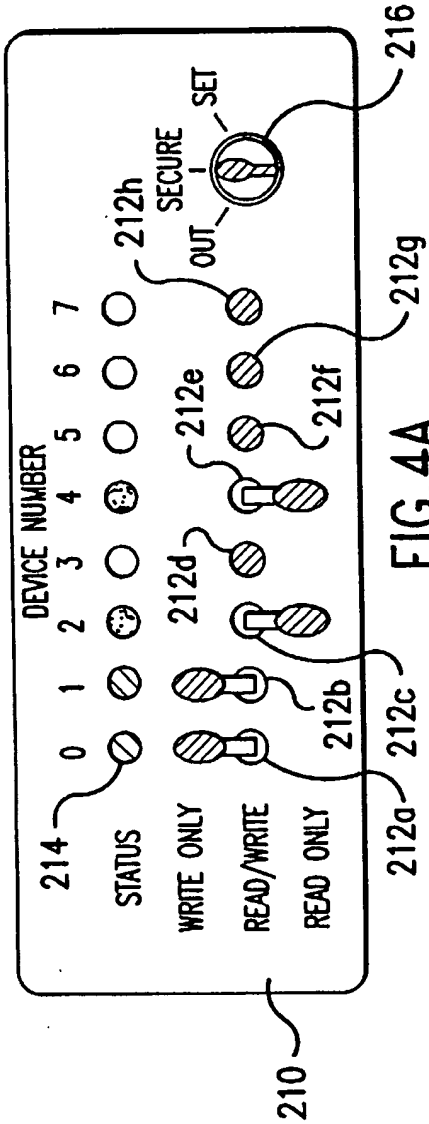

DEVICE NUMBER
0 1 2 3 4 5 6 7
STATUS
WRITE ONLY
READ/WRITE
READ ONLY
SECURE
OUT
SET
214
212a
212b
212c
212d
212e
212f
212g
212h
216
210

FIG.4A

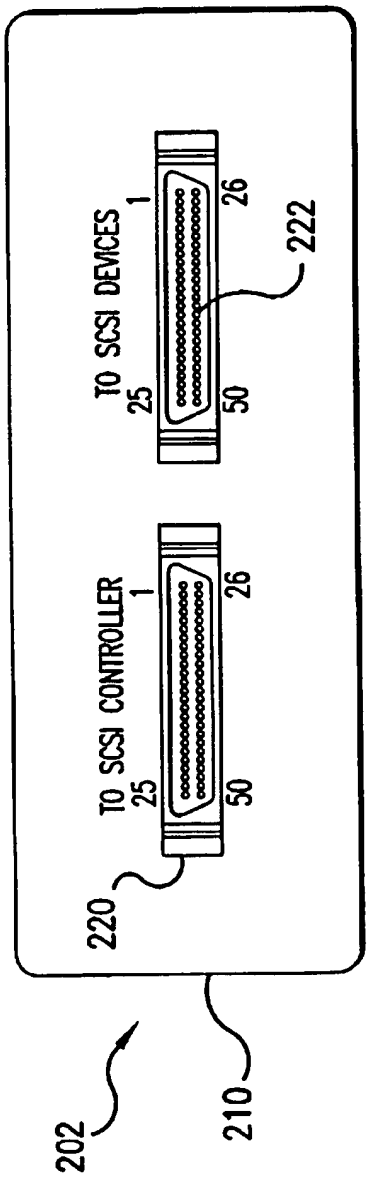

TO SCSI CONTROLLER
TO SCSI DEVICES
1
25
26
50
220
222
202
210

FIG.4B

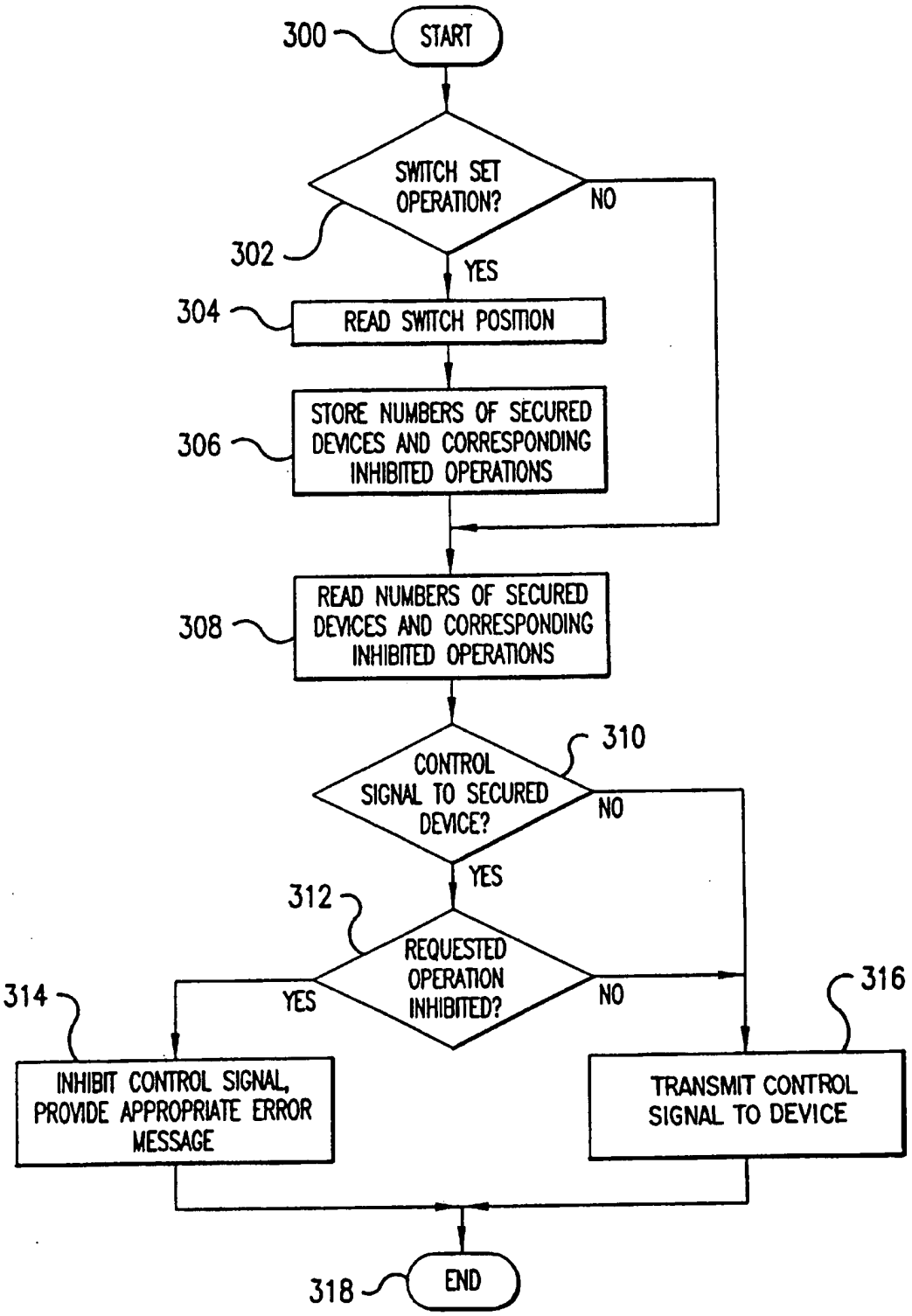

300
START
SWITCH SET OPERATION?
NO
302
YES
304
READ SWITCH POSITION
306
STORE NUMBERS OF SECURED DEVICES AND CORRESPONDING INHIBITED OPERATIONS
308
READ NUMBERS OF SECURED DEVICES AND CORRESPONDING INHIBITED OPERATIONS
310
CONTROL SIGNAL TO SECURED DEVICE?
NO
YES
312
REQUESTED OPERATION INHIBITED?
YES
NO
314
316
INHIBIT CONTROL SIGNAL, PROVIDE APPROPRIATE ERROR MESSAGE
TRANSMIT CONTROL SIGNAL TO DEVICE
318
END

FIG.5

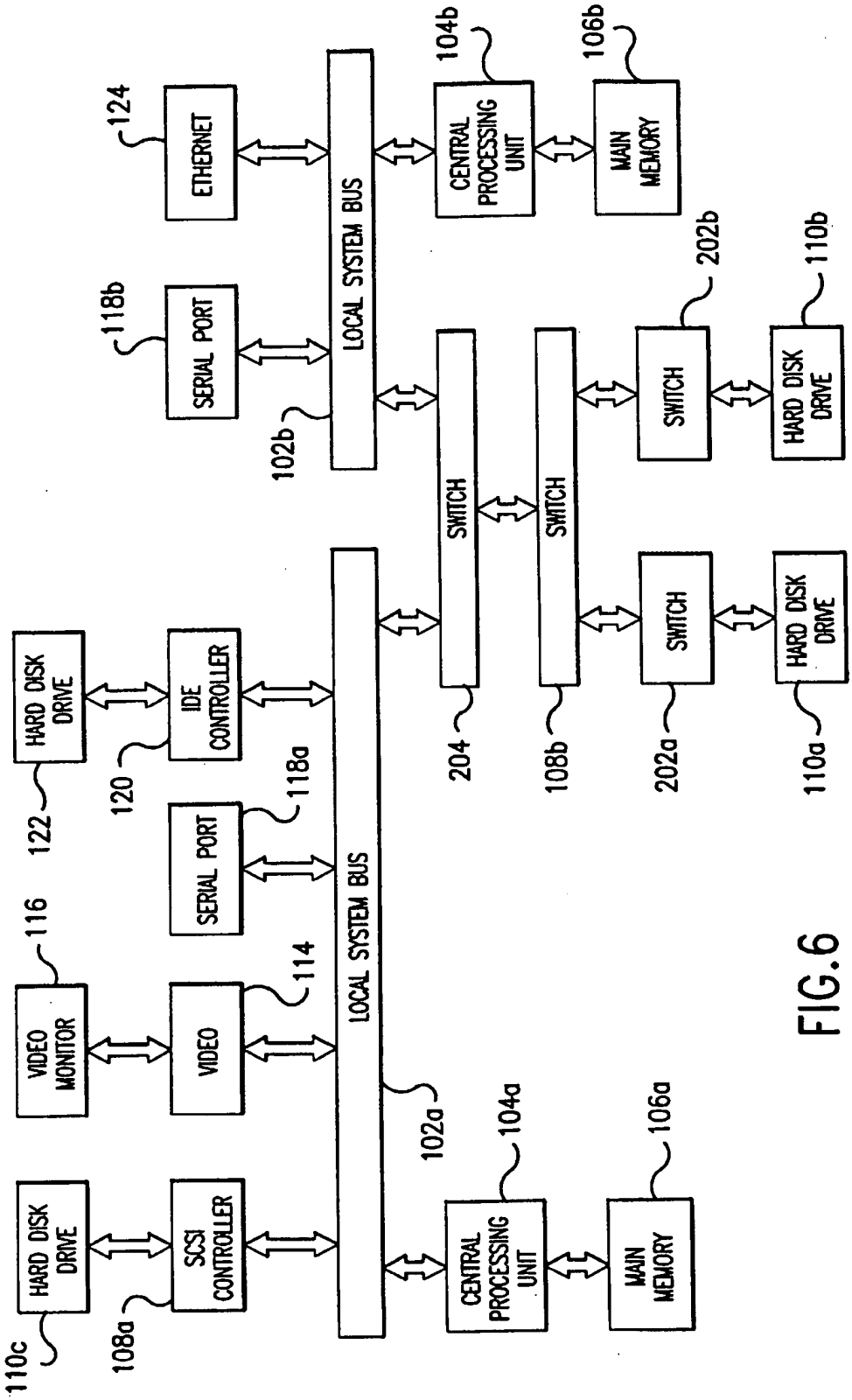

ETHERNET
124
LOCAL SYSTEM BUS
CENTRAL PROCESSING UNIT
104b
MAIN MEMORY
106b
SERIAL PORT
118b
102b
SWITCH
SWITCH
SWITCH
202b
HARD DISK DRIVE
110b
HARD DISK DRIVE
IDE CONTROLLER
120
122
118a
204
108b
202a
110a
SERIAL PORT
LOCAL SYSTEM BUS
VIDEO MONITOR
VIDEO
116
114
102a
HARD DISK DRIVE
SCSI CONTROLLER
CENTRAL PROCESSING UNIT
104a
MAIN MEMORY
106a
110c
108a

FIG.6

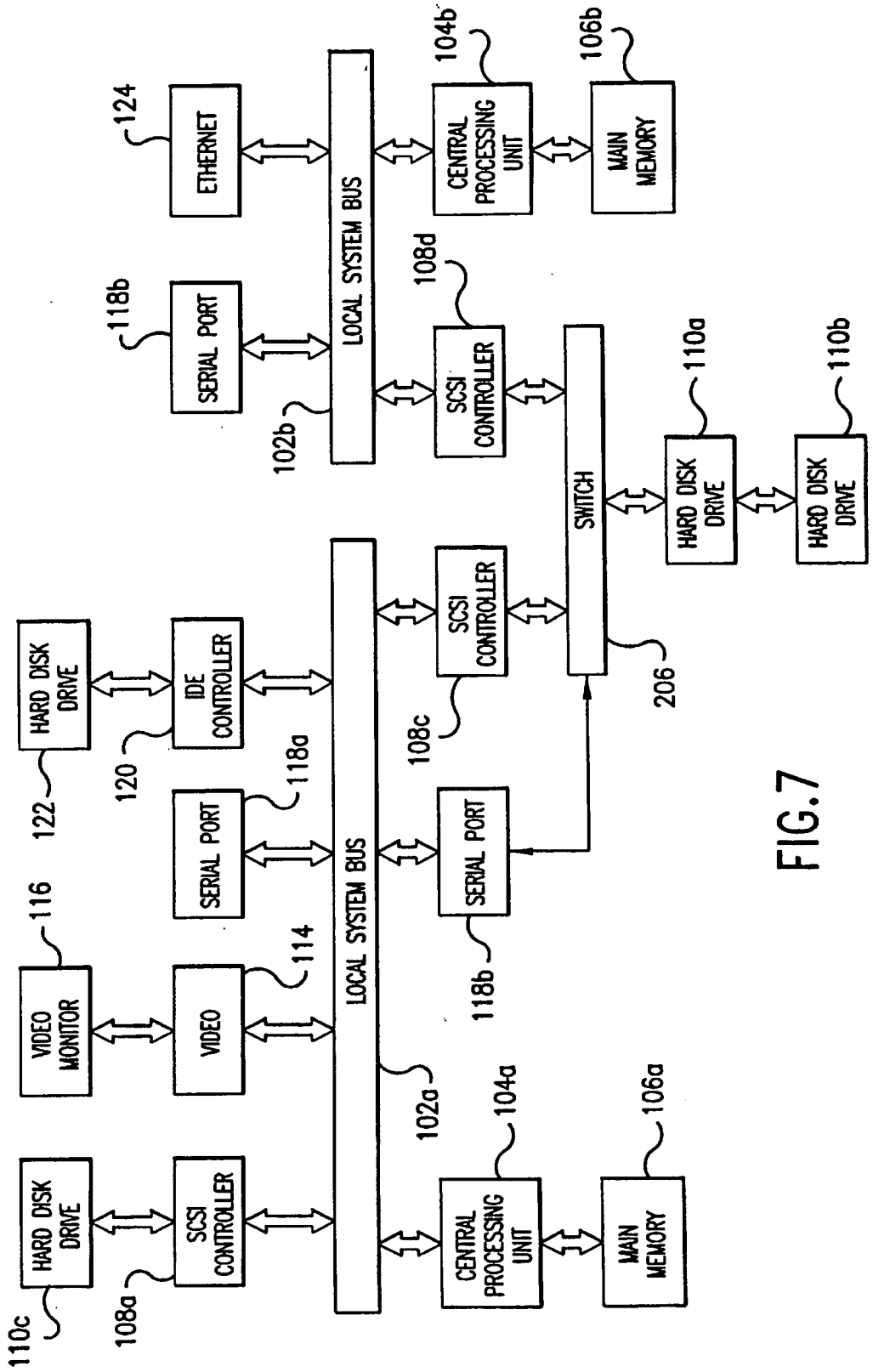

ETHERNET
124
LOCAL SYSTEM BUS
CENTRAL PROCESSING UNIT
104b
MAIN MEMORY
106b
SERIAL PORT
118b
102b
SCSI CONTROLLER
108d
SWITCH
HARD DISK DRIVE
110a
HARD DISK DRIVE
110b
206
HARD DISK DRIVE
122
IDE CONTROLLER
120
SERIAL PORT
118a
SCSI CONTROLLER
108c
LOCAL SYSTEM BUS
SERIAL PORT
118b
VIDEO MONITOR
116
VIDEO
114
102a
CENTRAL PROCESSING UNIT
104a
MAIN MEMORY
106a
HARD DISK DRIVE
110c
SCSI CONTROLLER
108a

FIG.7

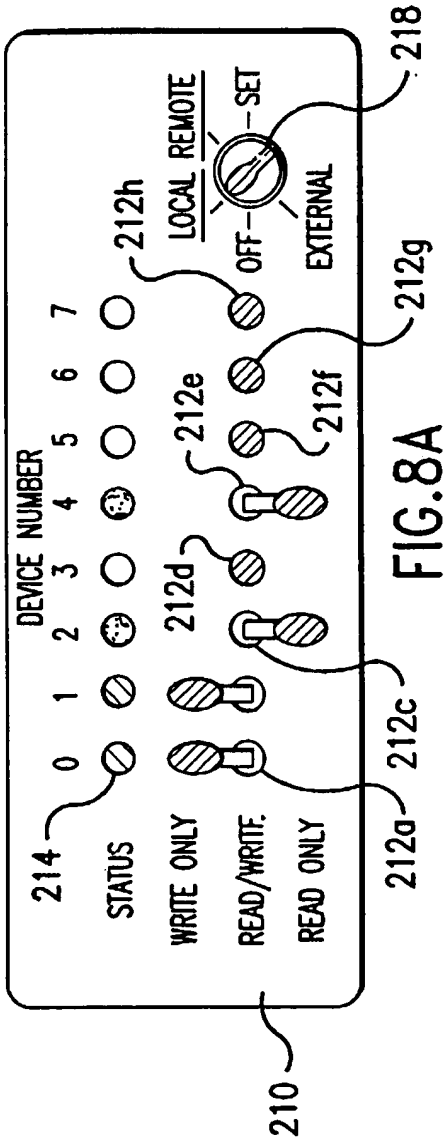

DEVICE NUMBER
0 1 2 3 4 5 6 7
214
STATUS
WRITE ONLY
READ/WRITE
READ ONLY
212a
212c
212d
212e
212f
212g
212h
LOCAL REMOTE
OFF
SET
EXTERNAL
218
210

FIG.8A

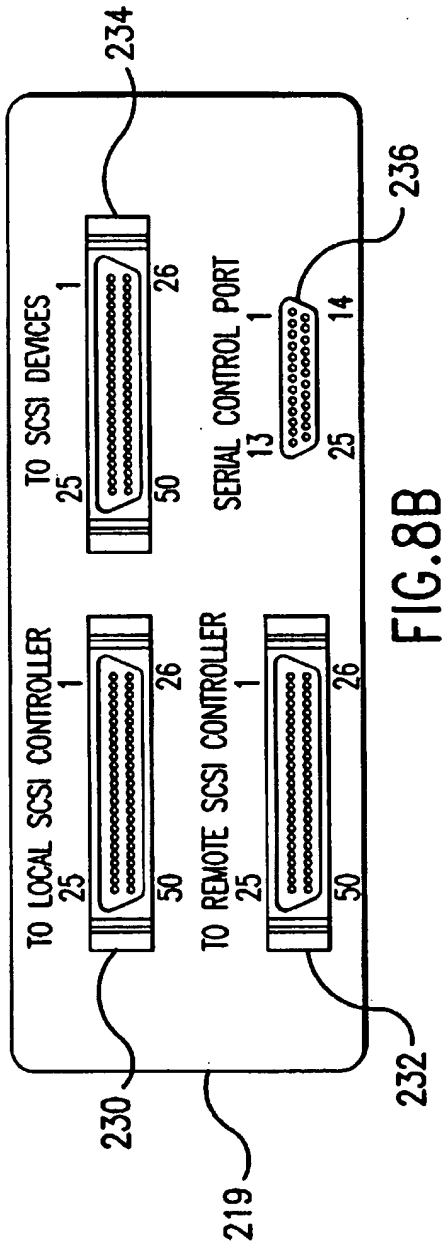

TO LOCAL SCSI CONTROLLER
TO REMOTE SCSI CONTROLLER
TO SCSI DEVICES
SERIAL CONTROL PORT
25
50
26
13
14
234
236
230
232
219

FIG.8B

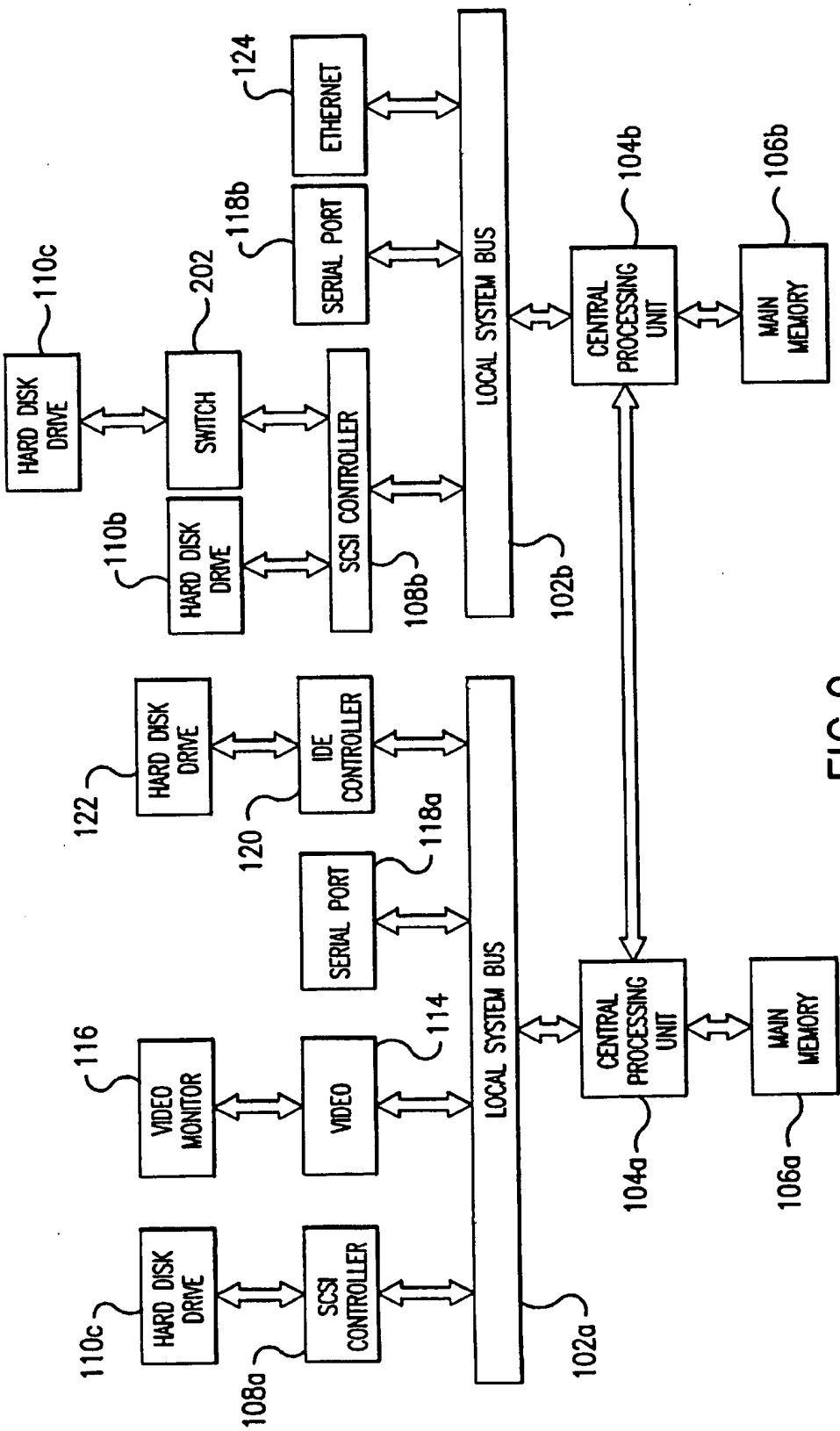

HARD DISK DRIVE
110c
SWITCH
202
SCSI CONTROLLER
108b
HARD DISK DRIVE
110b
SERIAL PORT
118b
ETHERNET
124
LOCAL SYSTEM BUS
102b
CENTRAL PROCESSING UNIT
104b
MAIN MEMORY
106b
HARD DISK DRIVE
122
IDE CONTROLLER
120
SERIAL PORT
118a
VIDEO MONITOR
116
VIDEO
114
HARD DISK DRIVE
110c
SCSI CONTROLLER
108a
LOCAL SYSTEM BUS
102a
CENTRAL PROCESSING UNIT
104a
MAIN MEMORY
106a

FIG.9

SECURE COMPUTER SYSTEM AND METHOD OF PROVIDING SECURE ACCESS TO A COMPUTER SYSTEM INCLUDING A STAND ALONE SWITCH OPERABLE TO INHIBIT DATA CORRUPTION ON A STORAGE DEVICE

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to computer system architecture and more particularly to an architecture for and method of limiting remote access to programs and data.

2. Description of the Related Technology

The role of computers is rapidly changing from computational machines to communication devices. The increasing use of the Internet by the general public increases the potential for hackers to break into sensitive computers. Computer hackers have successfully entered systems believed to be secure, gained unauthorized access, corrupted data, and infected systems with viruses that continue to cause havoc. While specialized software in the form of, for example, firewalls, is often provided to prevent unauthorized system access and to limit access so that unauthorized personnel cannot easily corrupt data and program files or otherwise cause damage to a computer system and loss of data, hackers are continually finding ways around the software. For example, viruses can be used to infect a computer system through infected software, causing the system to perform unauthorized functions and execute "rogue" code jeopardizing the integrity of the system. Because all functions performed by the computer system are controlled by instructions stored in the computer's memory, providing any remote access to the system provides an avenue for hackers to gain unauthorized access and do damage.

A representative computer system according to the prior art is shown in block diagram form in FIG. 1. A prior art computer system **100** includes a local system bus **102** connecting major elements of the computer system. Thus, local system bus **102** handles the transfer of instructions, data, address and control signals, etc. between the elements of the computer system. As shown in the figure, central processing unit **104** has a direct connection to bus **102** and to a dedicated main memory **106**. Main memory **106** is typically a high speed, high bandwidth random access memory storing data and instructions. Non-volatile mass storage is provided by hard disk drives **110** and **112** interfacing via SCSI (small computer systems interface) device **108** to local system bus **102** and hard disk drive **122** interfacing through IDE (intelligent drive electronics) controller **120**. Central processing unit **104** also has provisions for displaying data to a system operator by providing appropriate address, data and control signals to video interface **114** whereby data is displayed on video monitor **116**. Finally, remote access to peripheral devices and buses is provided by serial port **118** and Ethernet interface **124**, again over local system bus **102**. Although not shown, other devices providing input and output to the system may be included, such as a keyboard, etc., which may include a dedicated interface to local system bus **102** or might be supported by serial port **118**. Similarly, other output devices may be included, such as a printer interfacing through serial port **118** or an equivalent parallel port type data connection (not shown).

In operation, computer programs consisting of executable code and data and other information on which the code operates, are stored in main memory **106**. Typically, this

includes an operating system, such as Windows NT or Windows 98, together with various utilities and application programs. At startup or initialization, central processing unit **104** executes "boot" code, identifies system assets, such as IDE controller **120** and hard disk drive **122**, and locates the appropriate operating system. The operating system software from hard disk drive **122** is then transferred through IDE controller **120** via local bus **102** to main memory **106**. Central processing unit **104** then executes the operating system, transferring instructions as needed from main memory **106** into a "cache" or other local memory and registers that are a part of the central processing unit **104**. While this is happening, dedicated hardware and firmware resident in video board **114** provide a visual display on video monitor **116** of system status and provide a video output for the operating system, utilities, and application programs. In addition to the online data storage provided by hard disk drive **122**, multiple hard disk drives are supported by SCSI controller **108**. As depicted, both hard disk drives **110** and **112** are interfaced to local system bus **102** through the SCSI controller **108** providing additional non-volatile storage capabilities.

In addition to local access to computer system **100**, remote access is provided by serial port **118** and Ethernet card **124**. For example, a modem (not shown) may be attached to serial port **118** to interface computer system **100** to other media such as the public switched telephone network (PSTN), radio and fiber optic systems, etc., thereby providing connectivity to remote users and systems. An appropriate communications utility or application running on central processing unit **104** together with serial port **118** supports exchange of data with the remote users and systems. Similarly, Ethernet **124** is a specific embodiment of a network connectivity supporting, for example, a local area network (LAN), a wide area network (WAN), etc., with multiple remote computer systems and other resources attached. Using these remote access facilities, computer system **100** becomes accessible to authorized, and in many cases, unauthorized users.

Although not shown, other peripherals may be included, such as CD-ROMS (compact disk—read only memories), CD-WORM (compact disk—write once read many) or CD-WO (compact disk—write once), CD-RW (compact disk—re-writeable), DVD-RAM (digital versatile disk—RAM), DVD-ROM (digital versatile disk—ROM), various tape drives and traditional 3½ inch floppy disk drives. These devices are particularly useful for the transport of data between systems and backup purposes using removable media. Conventionally, because of access speed and storage space limitations, these devices are generally not relied upon as substitutes for hard disk drives which continue to be used as the primary media for non-volatile program and data mass storage. However, as computer systems have been made available to greater numbers of users, both locally and remotely, maintaining the integrity of programs and data stored on computer systems has become an increasing concern.

Prior art systems implement various physical and software systems to control access to the system and provide security. For example, computer systems handling classified information may require TEMPEST approval to avoid unintended radiation of information, be located in a secure facility such as a limited access area to provide physical security, and be operated in a stand alone configuration without provision for remote access to avoid remote hacker access. Physical security, however, cannot address remote access users so that a variety of software is used to establish

varying authorization levels for remote system use and access. For example, remote users may be required to interface via a secure access or "firewall" system which requires a user to establish authorization to access a computer system prior to providing a connection. A firewall may further monitor use of facilities, limiting access and use according to the user's authorization. Software on the computer system itself further monitors access using, for example, passwords, personal identification numbers (pins), etc. to control access and use. Other software may be implemented to protect, for example, certain area of memory such as the operating system from being altered or overwritten. Some operating systems, for example, further limit write operations to particular areas of memory containing data used by a particular application and limit access to other areas of memory or alteration of instructions stored in memory. However, such software protections have often proved inadequate to stop a determined hacker from gaining unauthorized access and bypassing such safeguards. For example, a hacker might use another program to generate and try thousands or millions of access code combinations to break into a system. Alternatively, using a more conventional approach, a hacker might rummage through discarded company documents to obtain access code information, unlisted maintenance telephone numbers, etc. Access may also be obtained by "back doors" into the system otherwise used for maintenance, billing, and other non-remote access purposes. Hackers may also obtain access by implanting computer viruses into the system, often embedded in innocent appearing host software. Once implanted, the virus can damage the system directly or provide other methods of access for the hacker.

In addition to remote covert action, computer systems are also subject to local attacks by, for example, disgruntled employees, etc. On a less sinister basis, computer systems are further subject to unintentional damage by human operator error inadvertently deleting or modifying files and by program bugs in the system and applications having similar effects and results as that of intentional attacks on the system.

For the foregoing reasons, there exists a need for a secure computer system architecture and method for providing computer security which cannot be easily bypassed by innocent or surreptitious means, either remotely or local to the computer system. A further need exists for a computer system and method of operating a computer system which preserves data and program integrity while providing for remote access to users having only read access. A still further need exists for a computer system and method of operating a computer system which prevents data and instruction corruption, modification and deletion by improper operation of host applications or due to the intentional actions of software viruses and other rogue executable code.

SUMMARY OF THE INVENTION

The present invention is directed to a computer system and method of operating a computer system which provides enhanced data and program security. A system and method according to the invention limit access to computer system storage media by providing a locally operable switch which selectively prevents alteration to the local storage media. The switch may be a manually operable mechanical device or may be electronic, so long as its operation is isolated from the system being protected, and may be entirely self-contained. For example, the appropriate control lines between a hard disk controller and the hard disk drive are routed through a manually operable electrical switch which can only be manipulated locally and cannot be operated or bypassed under computer control. In one configuration, the appropriate write enabling conductor of the cable is physically interrupted by the mechanical switch when in a secure mode and, instead, the appropriate write disabling signal is applied to the hard disk drive. This basic configuration and method can be applied to various computer system architectures to support stand alone, multiuser and remote access capable computer systems.

According to another aspect of the invention, a computer system includes dual processor elements, one isolated from remote access and having facilities for writing information to a storage device. The other processor element, while handling communications with remote devices, is connected so as to positively inhibit writing or altering data contained in the storage device. To further protect system integrity, another aspect of the invention configures the communications processing element as a slave, receiving and executing instructions from the isolated processing element. The invention further divides data storing and retrieval functions between a pair of hard disk drives used to provide remote access. Using this division, remote users may read from one hard disk drive, but are incapable of altering the contents of the read only drive. Similarly, remote users can write to the other hard drive, but cannot read information stored by other users and cannot target information for alteration or destruction.

According to an aspect of the invention, a digital computer system includes a processor, a storage device and a manually operative switch. The storage device is responsive to the processor for selectively operating in a read mode of operation for reading previously stored data and in a write mode of operation for storing data. The manually operative switch selectively disables the processor from causing the storage device to operate in the write mode of operation. According to a feature of the invention, the manually operative switch is connected to interrupt the control signal required to cause the storage device to operate in the write mode of operation. The manually operative switch may be in direct electrical contact with the storage device and may be in the form of a mechanical switch or may be an electronic switch including control software and hardware components.

According to another feature of the invention, the processor includes a central processing unit, a controller which is in direct electrical contact with the manually operative switch, and a bus which connects the central processing unit and the controller.

According to another aspect of the invention, a digital computer system includes a storage device, first and second central processing units, and a first manually operative switch. The storage device is responsive to a control signal for selectively operating in a read mode of operation for reading previously stored data and in a write mode of operation for storing data. The first and second central processing units are each capable of providing this control signal. The switch then alternatively provides the control signal from either the first or second central processing unit to the storage device. According to a feature of the invention, the system further includes a second manually operative switch selectively disabling the storage device from operating in the write mode of operation.

According to another aspect of the invention, a digital computer system includes a processor, a secure data storage device and a manually operative switch. The secure data

device is responsive to a write control signal from the data processor for selectively storing data. The switch is manually selectable to enable and disable receipt by the secure data storage device of the write control signal.

According to a feature of the invention, the manually operative switch selectively applies a predetermined fixed control signal to the secure data storage device instead of the write control signal. The secure data device may be anon-volatile memory including a hard disk drive.

According to another feature of the invention, a bus connects the processor to the secure storage device for transmission of the control signal so that the manually operative switch selectively enables and disables a transmission of the control signal along the bus.

According to another feature of the invention, the processor includes a central processing unit and a disk controller connected to each other by a system bus. The secure data device includes a disk drive electrically connected through the manually operative switch to the disk controller for receiving the control signal so that the manually operative switch selectively enables and disables transmission of the control signal. Another disk drive may be included together with another disk controller connected to the system bus for selectively writing data to and reading data from the additional disk drive in the form of, for example, an array of multiple hard disk drives (e.g., redundant array of independent disks, or "RAID"). These additional disk drives may be connected independent of the manually operative switch or may be connected with a second manually operative switch to prevent writing to the additional disks.

According to another feature of the invention, the digital computer system further includes first and second disk controllers connected to respective master and slave central processing units by a system bus. The secure data storage device includes a first disk drive electrically connected through the manually operative switch to the first disk controller for receiving a control signal from the master central processing unit whereby the manually operative switch selectively enables and disables transmission of the control signal to the first disk drive. The second disk drive is connected to the second disk controller and is accessible by the master and slave central processing units over the system bus. Alternatively, the first and second disk controllers may be included on separate buses accessible only by the respective master and slave central processing units.

According to another feature of the invention, a second manually operative switch is interposed between the second disk drive and the second disk controller to selectively disable reading from or, in an alternate configuration, writing to the second disk drive.

According to another feature in the invention, the computer includes a third disk controller and disk drive with the disk drive operative to mirror data stored in the second disk drive.

According to another feature in the invention, the computer system includes a first program memory connected to and storing instructions executable by the master central processing unit. A second program memory is connected to and stores instructions executable by the slave processing unit with a processor bus connecting the master and slave central processing units. A communications controller may be connected to the system bus to provide for remote access.

According to another aspect of the invention, a computer system includes a processor, a manual switch and a data storage device. The switch is connected to selectively transmit a control signal received from the processor and, alternatively, a write inhibiting control signal. In response to the signal received from the switch, the data storage device selectively stores data or is inhibited from doing so.

According to a feature of the invention, the storage device is responsive to the control signal transmitted by the manual switch for selectively operating in read and write modes of operation so that the write-inhibiting control signal causes the data storage device to operate only in the read mode of operation and/or other modes protecting the integrity of the data (e.g., internal refresh only).

According to another feature of the invention, the processor includes a first disk controller and the data storage device is a first disk drive. According to another feature of the invention, a second disk drive may also be connected to the first disk controller or may be connected to its own, second disk controller.

According to another aspect of the invention a digital computer system includes a processor, a storage device and a switch. The storage device is responsive to the processor for selectively operating in a plurality of operating modes including a read mode of operation for retrieving previously stored data and a write mode of operation for storing data. The switch is operable to selectively enable and disable at least one of the operating modes, the switch being controllable by means distinct and separate from the processor so that the processor is inhibited from controlling the operation of the switch. According to a feature of the invention, the switch may be manually operated to selectively make and break an electrical conducting path connecting the processor with the storage device.

Alternatively, the switch may include a controller, an operation of which is independent of the processor for selectively enabling and disabling at least one of the operating modes. At least one of the operating modes may be a read mode of operation and, alternatively, may be a write mode of operation. According to a feature of the invention, a second "master" processor is isolated from the first processor and both (i) controls the switch and (ii) reads and writes to the storage device.

According to another feature of the invention, the storage device may include a magnetic media and comprise a disk drive or a magnetic tape. The storage device may alternatively include a non-volatile electronic memory device, such as an EEPROM.

According to still a further feature, the storage device may include an optical storage device such as a CD-ROM or an electro-optical source device such as a CD-RW.

According to still another feature of the invention, the digital computer includes a processor with a first memory storing program instructions and a distinct and separate memory storing data. The first memory may be operable in the read only mode of operation in which the program instructions are protected from alteration and erasure by the central processing unit.

According to another aspect of the invention, a method of operating a digital computer system includes the steps of supplying a variable control signal to a disk drive and writing data to the disk drive in response to the variable control signal. A manual electrical switch is operated so as to disconnect the variable control signal from the disk drive and instead, connect a fixed control signal to the disk drive. The disk drive is then operated in a mode other than a write mode of operation in response to the fixed control signal. According to a feature of the method, remote access to the disk drive is provided only when operating in the mode other than the write mode of operation, i.e., in the secure mode inhibiting changes to the hard disk drive.

These and other features, aspects and advantages of the present invention will become better understood with regard to the following description, claims and accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of a computer system according to the prior art.

FIG. **2** is a block diagram of a computer system according to the invention including a switch for inhibiting a hard disk drive from operating in a write mode of operation and segmented main memory.

FIG. **3** is a pin-out diagram and table for an IDE connector.

FIGS. **4***a* and **4***b* are front and rear views of a stand alone switch device for insertion between a SCSI controller and one or more SCSI devices.

FIG. **5** is a flow diagram for a software implemented switch for restricting operation of designated peripheral devices to programmed modes of operation.

FIG. **6** is a block diagram of a computer system according to another embodiment of the invention including a switch for connecting a storage unit to a stand alone processing unit or to a processor providing for remote access.

FIG. **7** is a block diagram of a computer system according to another embodiment including isolated (i) secure local and (ii) remote processing systems sharing common hard disk facilities under the exclusive control of the secure local processor.

FIGS. **8***a* and **8***b* are front and rear views of a switching device for selectively connecting one of two SCSI controllers to a plurality of SCSI devices and for limiting operation of those SCSI devices to programmed modes of operation when connected to the second of the SCSI controllers.

FIG. **9** is a block diagram of a computer system according to the invention including a master/slave architecture.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Referring to FIG. **2** of the drawings, a computer system **200** includes conventional devices **102–124** as discussed in connection with the prior art with the (i) addition of switch **202** interposed between SCSI controller **108** and hard disk drive **110** and (ii) partitioning of main memory into separate instruction memory **106***a* and data memory **106***b*. Instruction memory **106***a* may include various forms and levels of protection. For example, instruction memory **106***a* may be implemented in the form of an EEPROM with a manual erase and programming feature. Thus, CPU **104** would have read-only access to instruction memory **106***a* unless and until the associated EEPROM was manually provided with the proper control signals to allow its programming. This feature prevents unauthorized modification of programming and provides security against viruses attacking the program code. In contrast, data memory **106***b* is a conventional RAM for the temporary storage of data, including system and application program parameters and variables.

Switch **202** may be configured as a part of SCSI controller **108**, hard disk drive **110**, or as a separate auxiliary device. Switch **202** may be exclusively manually operable to inhibit a hard disk drive from altering or erasing data. Alternatively, switch **202** may be an electronic switch controlled by a control signal physically inaccessible to or by CPU **104**. Typically, hard disk drive **110** is responsive to read and write requests from SCSI controller **108**. Switch **202** is effective to selectively inhibit operation in the write (or read) mode so that, effectively, hard disk drive **110** can be operated in either a read/write mode, or in a read only or write only mode of operation.

If switch **202** is included as part of SCSI controller **108**, then it is connected to inhibit write requests from CPU **104** (or other devices) from being sent to hard drive **110**. If switch **202** is instead incorporated into hard drive **110**, it can be connected to inhibit operation of hardware used to operate the disk drive's write heads. For example, the switch **202** can be configured to cut power to a write head's output circuitry. Preferably, hard disk drive **110** and/or SCSI controller **108** provide the appropriate status and/or error messages to CPU **104** when operating in a write inhibited or read only mode of operation or when a write operation is requested and the write mode has been disabled.

Switch **202** may also be configured as an auxiliary, stand alone device mounted in a switch box enclosure with appropriate terminals for connecting controller **108** to hard disk drive **110**. In this configuration, switch **202** is operative in a first read/write position to pass signals from controller **108** to hard disk drive **110** without change. In a write inhibit or read only mode of operation, switch **202** will not pass signals from controller **108** to hard disk drive **110** which would cause hard disk drive **110** to be placed in a write mode of operation. For example, pin **50** of a SCSI interface may be set to the appropriate logic level when a selected device is accessed so as to limit operation of the selected device to either a read or write mode as appropriate. Alternatively, switch **202** may be connected between IDE controller **120** an hard disk drive **122** to selectively restrict access and control of the latter. Using an IDE interface, a pin-out diagram for which is shown in FIG. **3** of the drawings, write strobes from the controller are transmitted to the hard drive on pin **23**. That is, the controller signals the hard drive that data supplied on pins **3–18** is ready to be written by driving a control signal applied at pin **23** to a "low" logic level. Thus, in a secure mode of operation wherein writing to a hard drive is to be inhibited, pin **23** is connected to a high level logic signal source so that the hard disk drive does not receive the write strobe signal necessary to cause it to perform a write operation.

Alternatively, switch **202** may include appropriate hardware and software to monitor signals transmitted by controller **108** to hard disk drive **110**. Write (or other inhibited actions such as read, erase, etc.) commands to one or more designated devices would be recognized and intercepted, switch **202** generating an appropriate error message back to controller **108**. Permissible operations would be transmitted through to disk drive **110** without impediment. In this software implementation of switch **202**, predetermined portions of disk drive **202** may be designated as secure so that write commands are selectively inhibited only to designated tracks, sectors, clusters, etc.

FIGS. **4***a* and **4***b* show a stand alone, programmable embodiment of switch **202** which can accommodate eight peripheral devices on a SCSI interface. Switch **202** is mounted in enclosure **210** and includes panel mounted programming switches **212***a–212h* associated with respective SCSI devices **0–7**. Each of the programming switches is selectable to designate a read only, read/write, or write only mode of operation for the respective device. Once programmed, the status of each device is indicated by a tricolor LED **214** associated with each switch, green, for example, indicating read/write capabilities, yellow that the corresponding device can be operated in a read only mode of operation (write-inhibited), and red indicating that the

corresponding device is operable in a write only mode of operation (i.e., read operations are inhibited). As shown in FIG. 4*a*, devices **0** and **1** are being operated in write only modes (i.e., a "secure" mode), devices **2** and **4** in read only modes (another "secure" mode), and devices **3**, **5**, **6**, and **7** in read/write modes (i.e., are not being operated as "secure" devices).

A key switch **216** may be included to control the operation of switch **202**. In the "OUT" mode, the switch is functionally inoperative so that the operations of all devices are unrestricted as would be indicated by green status lights **214**. In the "SECURE" mode, the programmed mode limits would be effective to limit read and write modes of operations. The "SET" mode is used to program switch **202** according to switches **212***a*–**212***h*. A corresponding key (not shown) is removable from key switch **216** in the "OUT" and "SECURE" positions so that switch **202** can be left locked and unattended. Preferably, the "SET" position of key switch **216** is a temporary position with a spring returned to the "SECURE" position upon completion of programming. When switch **216** is in the "SET" mode, the position of switches **212***a*–**212***h* are read and the corresponding mode limitations are stored in memory as would be indicated by status indicators **214**.

A rear view of switch **202** is presented in FIG. 4*b* including panel mounted SCSI connectors **220** and **222** for connecting the switch to a SCSI controller and to SCSI devices being controlled, respectively.

The operation of switch **202** is shown in the flow diagram of FIG. **5**. The program starts at entry point **300** with an initial decision box **302** handling the set mode of operation for programming the device. If switch **202** is in the "SET" mode, then the positions of switches **212***a*–**212***h* are read at process **304** and the corresponding limitations are stored in memory at process **306**. If the SET operation has not been activated, or upon completion of the programming, processing continues at step **308** where the numbers of the secure devices are read from memory together with the corresponding allowed modes or inhibited modes of operations, as appropriate. In response to receipt of a control signal at decision **310**, the program decides if the control signal is directed to a secure device, i.e., a device number previously stored as being operated in a "SECURE" mode with either read or write operations inhibited. If the control signal is directed to a device which is not subject to read or write limitations, such as devices **3**, **5**, **6**, and **7** according to FIG. 4*a*, then the control signal is transmitted to that device at process **316**. However, if the control signal is directed to a device which is being operated in a secure mode of operation (devices **0**, **1**, **2**, and **4** in this example), then the process determines at decision box **312** if the requested operation has been inhibited. For example, device numbers **0** and **1** are being operated in a read-inhibited mode while devices **2** and **4** are being operated in a write-inhibited mode. Accordingly, read requests directed to devices **0** or **1** and write requests directed to devices **2** and **4** would result in the left branch being taken out of decision point **312** where the appropriate control signal would be inhibited and an error message generated back to the requesting controller. Conversely, if the operation requested has not been inhibited, the right branch is taken out of decision box **312** and the request is transmitted to the device address. In either case, process flow continues down to terminal **318**. At this point, the process would conventionally loop back to Start **300** to continuously detect and process programming requests and SCSI interface commands.

Another embodiment of the invention is shown in FIG. **6** depicting a dual processor system, with both read/write and read only hard drives, each having a dedicated bus, local memory and storage. A hard drive storage system is switchable between the processors. The hard drive storage system includes two disk drives, operable in a non-secure normal mode of operation in which both drives are read/write enabled, and in a protected mode wherein one drive is operated in a read only mode and the other in a write only mode of operation. In this configuration, the two processors are isolated from each other, one of the processors providing for local system operation, the other providing remote access to the mass storage devices including hard disk drives. In effect, the system is equivalent to two separate independent systems on one motherboard when configured as a personal computer (PC). Both systems require software to be loaded, and some system configuration to be performed. Communications between the processors is provided by the common hard drive storage system.

Operator monitoring of the system performance and downloading of data acquired by the system is performed by a primary CPU **104***a* connected to a first local system bus **102***a*. The second local data bus **102***b* supports a communications CPU **104***b*. Connected to both buses **102***a* and **102***b*, switch **204** physically switches SCSI controller **108***b* between the two buses. Hard disk drives **110***a* and **110***b* are connected and controlled by SCSI controller **108***b* through write mode disabling switch **202***a* and read mode disability switch **202***b*, respectively. Switchable SCSI controller **108***b* would be switched to main local system bus **102***a* for loading and configuration of software under control of main CPU **104***a*. After loading and testing of software, SCSI controller **108***b* would be switched to local system bus **102***b* supporting communications with remote users over serial port **118***b* and Ethernet **124**. Hard disk drive **110***a* would be then operated in a read only mode of operation by switch **202***a*. Conversely, hard disk drive **110***b* would be operated in a "write only" mode of operation so that, for example, any uploaded data could be checked for viruses prior to that data becoming available to the system. Further, by placing hard disk drive **110***b* in a "write only" mode of operation using switch **202***b*, data uploaded to the drive by remote users of the system cannot be accessed by other remote users thereby enhancing system security. This feature is particularly useful for e-commerce applications where confidential data received from remote user must be protected from unauthorized dissemination (e.g., credit card information, etc.).

In the configuration of FIG. **6**, the primary CPU **104***a* and associated first bus **102***a* are inaccessible to remote users. Accordingly, switch **204** and switches **202***a* and **202***b* may be electronically controlled by primary CPU **104***a* without jeopardizing the security of the system. This feature is incorporated into the configuration shown in FIG. **7** wherein SCSI controllers **108***c* and **108***d* are connected to respective first and second buses **102***a* and **102***b*. Switch **206** is controlled by CPU **104***a* via serial port **118***b* connected to first bus **102***a*. Switch **206** selectively connects either SCSI controller **108***c* or **108***d* to SCSI hard disk drives **110***a* and **110***b*.

In a local mode of operation, switch **206** provides unlimited access by local SCSI controller **108***c* to hard disk drives **110***a* and **110***b*. Thus, CPU **104***a* can both read from and write to the drives. Upon being commanded to connect the drives to second bus **102***b* to support remote access, switch **206** disconnects SCSI controller **108***c* and connects SCSI controller **108***d* to the drives subject to preprogrammed operating mode limitations. For example, when being accessed by SCSI controller **108***d*, hard disk drive **110***a* may be write inhibited while hard disk drive **110***b* may be read inhibited as described in connection with the configuration of FIG. **6**.

FIGS. **8***a* and **8***b* show an alternate implementation of a stand alone switch **210** suited to the dual processor system shown in FIG. **7**. The output of SCSI controller **108***c*, which is connected to local system bus **102***a*, is provided to connector **230** while SCSI controller **108***d*, which is connected to local system bus **102***b*, is connected to connector **232**. A serial connector **236** provides an interface for optional computer control of the switch.

In this configuration, switch **210** both switches hard disk drives **110***a* and **110***b* between the appropriate SCSI controller and selectively operates the hard disk drives in the pre-programmed restricted modes of operation. As shown, key switch **218** has five positions including "EXTERNAL", "OFF", "LOCAL", "REMOTE", and "SET". In the "OFF" mode, neither of the SCSI controllers have access to peripheral devices including the hard disk drives. In the "LOCAL" position, signals from and to connector **230** are passed through without alteration to SCSI devices connected to connector **234**. This mode is applicable to unrestricted operation of the peripheral devices when under control of primary CPU **104***a* which is inaccessible by remote users.

When key switch **218** is placed in the "REMOTE" position, connector **232** provides access to SCSI devices connected at connector **234** under the control and supervision of switch **210** to selectively inhibit predetermined modes of operation according to stored programming and as indicated by status indicator lights **214**. As previously described, a temporary, spring loaded "SET" position is provided for programming switch **210** according to the positions of switches **212***a*–**212***h*.

The "EXTERNAL" position allows a secure device, such as primary CPU **104***a*, to program and control switch **204** via a serial RS-232 interface. Thus, so long as the security of primary CPU **104***a* is not breached, the operating integrity of switch **202** is maintained.

Another embodiment in the invention including dual processors in a master/slave relationship is shown in the block diagram of FIG. **9**. According to this embodiment, one processor manages communications including, for example, responding to requests from the Internet. However, the slave processor only accepts program instructions from the primary processor. This can be accomplished by appropriate programming of the system firmware (e.g., BIOS) of the slave processor. Thus, the slave processor is controlled only by the master processor and would not be accessible by a remote computer hacker.

Referring to FIG. **9**, a master central processing unit **104***a* is connected to dedicated main memory **106***a* including an operating system. Master central processing unit **104***a* is connected via local system bus **102***a* to various devices including (1) hard disk drive **110***a* through SCSI controller **108***a*; (2) video control board **114** and video monitor **116**; (3) serial port **118***a*; and (4) hard disk drive **122** through IDE controller **120**. Slave central processing unit **104***b* provides remote access functions and is connected to a local main memory **106***b*. Central processing unit **104***b* connects to SCSI controller **108***b*, serial port **118***b* and Ethernet **124** through local system bus **102***b*. In turn, SCSI controller **108***b* connects to hard disk drive **110***b* and, via selectable "read only" switch **202**, to hard disk drive **110***c*. As previously mentioned, slave central processing unit **104***b* obtains operating instructions exclusively from master central processing unit **104***a* so that viruses or other changes cannot be remotely made to its operating instructions or programming. Critical data that is to be protected from change or deletion by remote users is stored in hard disk drive **110***c* operated in a read only mode of operation. Hard disk drive **110***b* supports storage of data by remote users, such as required for e-commerce, etc.

According to the invention as illustrated by the embodiments described, the capability of writing to and altering data is disabled for remote users by disabling hard disk write capabilities, limiting remote user access to a dedicated and segregated data processor and associated bus and data storage, and by isolating control of a communications processor so that instructions are only executed as received from a secured master processor. The invention further enhances security by segregating read and write functions to different hard drives so that remote users cannot alter information previously stored on the system nor can they read information stored by other remote users.

Although the present invention has been described in considerable detail with reference to certain preferred embodiments thereof, other embodiments or configurations are possible. For example, the mode limiting switch is applicable to other storage devices and media and to other devices where selection and control of operating modes must be restricted. For example, a restricted user may be limited by the switch to monitoring the output of a device such as a video camera, while a local user may additionally control the camera. Similarly, the switch may be used in-line with a printer to allow limited printing capabilities for certain users while providing full capabilities to local users of the system. Accordingly, the spirit and scope of the appended claims should not be limited to the description of the preferred embodiments contained herein.

What is claimed is:

1. A digital computer system comprising:

first and second electrically isolated buses;

first and second independent central processing units connected to a respective one of said first and second buses;

a storage device connected to each of said buses for selectively storing data; and

a manually operative switch selectively controlling access by said first central processing unit to inhibit storing data to said storage device by said first central processing unit without inhibiting storing data by said second central processing unit.

2. The digital computer system according to claim **1** wherein said storage device is operable in (i) a read mode of operation for reading previously stored data and (ii) a write mode of operation for storing said data.

3. The digital computer system according to claim **2** wherein said manually operative switch is connected to both said first and second buses to selectively operate said storage device in a write-only protected mode of operation.

4. The digital computer system according to claim **1** further comprising an interprocessor bus, said first central processing unit comprising a master central processing unit and said second central processing unit comprising a slave central processing unit, said master and slave central processing units connected to each other by said interprocessor bus and connecting to respective ones of said first and second buses, said manually operative switch connected to both said first and second buses and connected to selectively transmit to said storage device a control signal requited to cause said storage device to operate in said write mode of operation.

5. A digital computer system comprising:

first and second independent local buses;

first and second storage devices, each responsive to a control signal for selectively operating in (i) a read

mode of operation for reading previously stored data and (ii) a write mode of operation for storing data;

first and second central processing units respectively connected to said first and second local buses, each of said first and second central processing units capable of providing said control signal;

a first manually operative switch alternatively providing said control signals from said first and second local buses to said first and second storage devices, said switch further configured to selectively operate said first and second storage devices in a protected mode of operation, said protected mode of operation including at least one of a write-only and read-only mode of operation.

6. The digital computer system according to claim 5 further comprising a second manually operative switch selectively disabling at least one of said first and second storage devices from operating in said write mode of operation.

7. The digital computer system according to claim 5 farther comprising second and third switches, said second switch selectively inhibiting said first storage device from operating in said write mode of operation, said third switch selectively inhibiting said second storage device from operating in said read mode of operation.

8. The digital computer system according to claim 7 further comprising a communications interface providing remote access to said second local bus.

9. The digital computer system according to claim 5 further comprising switching means having a first state wherein said first and second storage devices are operable in both said read and write modes of operation and a second state inhibiting operation of said first storage device in said write mode and of said second storage device in said read mode.

10. The digital computer system according to claim 9 further comprising a communications interface providing remote access to said second central processing unit.

11. The digital computer system according to claim 5 further comprising switching means having a first state wherein said first and second disk storage devices are operable in both said read and write modes and a second state causing said first storage device to be operated only in said read mode of operation and said second storage device only in said write mode of operation.

12. The digital computer system according to claim 11 further comprising a communications interface providing remote access to said second central processing unit.

13. A digital computer system comprising:

first and second system buses electrically independent of each other;

master and slave central processing units connected to respective ones of said system buses;

first and second controllers respectively connected to said master and slave central processing units by respective ones of said system buses;

a data storage device responsive to a write control signal from one of said master and slave processing units on a respective one of said first and second system buses for selectively storing data said data storage device including first and second storage devices; and

a manually operative switch selectively enabling and disabling receipt by said data storage device of said write control signal from said first and second system buses.

14. The digital computer system according to claim 13 wherein said manually operative switch selectively connects said data storage device to one of said first and second controllers.

15. The digital computer system according to claim 13 wherein said manually operative switch is operative to selectively cause said data storage device to operate in a protected mode including a read-only and a write-only mode of operation independent of a mode control signal provided by one of said master and slave central processing units.

16. The digital computer system according to claim 13 wherein said manually operative switch is operative to selectively cause said data storage device to operate in a data protected mode including one of a read-only and write-only mode of operation independent of a mode control signal provided by one of said master and slave central processing units.

17. The digital computer system according to claim 13 further comprising a bus connecting one of said master and slave central processing units to said data storage device for transmission of said control signal wherein said manually operative switch selectively enables and disables a transmission of said control signal along one of said first and second buses.

18. The digital computer system according to claim 17 wherein said data storage device comprises a hard disk drive.

19. A digital computer system comprising:

a first data processing unit including a first central processing unit and a first disk controller connected to each other by a first system bus;

a second data processing unit including a second central processing unit and a second disk controller connected to each other by a second system bus, said second system bus electrically independent of said first system bus;

a secure data storage device responsive to a write control signal from each of said first and second data processing units for selectively storing data, said secure data storage device comprising a first disk drive; and

a manually operative switch selectively enabling and disabling receipt by said secure data storage device of said write control signal.

20. The digital computer system according to claim 19 wherein said first disk drive comprises an array of hard disk drives.

21. The digital computer system according to claim 19 further comprising another disk drive connected to one of said first and second disk controllers independent of said manually operative switch.

22. The digital computer system according to claim 19 wherein said first disk drive is electrically connected through said manually operative switch to said first disk controller for receiving said control signal whereby said manually operative switch selectively enables and disables a transmission of said control signal,

said digital computer system further comprising a second disk drive; and a second disk controller connected to said second system bus and to said second disk drive for selectively writing data to and reading data from said second disk drive.

23. A digital computer system comprising:

master and slave central processing units;

master and slave system buses electrically isolated from each other and respectively connected to said master and slave central processing units;

a secure data storage device responsive to a write control signal from each said master and slave central processing units for selectively storing data;

a manually operative switch configured to selectively enable and disable receipt by said secure data storage

device of said write control signal so as to selectively operate said secure data storage device in a read-only mode of operation; and

first and second disk controllers connected to said master and slave system buses, said secure data storage device including a first disk drive electrically connected through said manually operative switch to said first and second disk controllers for receiving said write control signal from one of said master and slave central processing units whereby said manually operative switch selectively enables and disables transmission of said write control signal.

24. The digital computer system according to claim **23** further comprising a second disk drive connected to said second disk controller.

25. The digital computer system according to claim **23** further comprising:

a first program memory connected to and stoning instructions executable by said master central processing unit,

a second program memory connected to and storing instructions executable by said slave central processing unit, and

a processor bus connecting said master and slave central processing units.

26. The digital computer system according to claim **23** further comprising a communications controller connected to said slave system bus.

27. A digital computer system comprising:

a first central processing unit;

a first system bus connected to said first central processing unit;

a second central processing unit;

a second bus connected to said second central processing unit and centrically isolated from said first system bus;

a disk controller;

a first manual switch selectively providing a conductive path between said disk controller and, in a first position, said first system bus and, in a second position, said second system bus; and

a hard disk drive connected to said disk controller and responsive to a write control signal from said disk controller for selectively storing information.

28. The digital computer system according to claim **27** further comprising a second manual switch interposed between said disk controller and said hard disk drive for selectively transmitting said write control signal therebetween so as to selectively permit an operation of said hard drive in a read-only mode of operation.

29. A digital computer system comprising:

a first system bus;

a second system bus

a first processor connected to said first system bus;

a second processor connected to said second system bus;

a data storage device connected to said first and second system buses for selectively operating in a plurality of operating modes so as to access said data storage device; and

a switch operable to selectively enable and disable at least one of said operating modes, said switch controllable by means distinct and separate from at least one of said processors whereby said one processor is inhibited from controlling said operation of said switch.

30. The digital computer system according to claim **29** wherein said switch comprises a manually operated switch connected to selectively make and break an electrically conducting path connecting of said first and second system base one processor and said data storage device.

31. The digital computer system according to claim **29** wherein said switch comprises a digital controller, an operation of which is independent of said second processor for selectively enabling and disabling said at least one of said operating modes.

32. The digital computer system according to claim **29** wherein said data storage device is operable in (i) a read-only mode of operation for retrieving previously stored data and (ii) a write-only mode of operation for storing data.

33. The digital computer system according to claim **32** wherein said at least one of said operating modes is said read-only mode of operation.

34. The digital computer system according to claim **32** wherein said at least one of said operating modes is said write-only mode of operation.

35. The digital computer according to claim **32** wherein said data storage device comprises a magnetic media.

36. The digital computer according to claim **32** wherein said data storage device comprises a disk drive.

37. The digital computer according to claim **32** wherein said data storage device comprises a magnetic tape.

38. The digital computer according to claim **32** wherein said data storage device comprises a non-volatile electronic memory device.

39. The digital computer according to claim **38** wherein said electronic non-volatile electronic memory device comprises an EEPROM.

40. The digital computer according to claim **32** wherein said data storage device comprises an optical storage device.

41. The digital computer according to claim **32** wherein said data storage device comprises an electro-optical storage device.

42. The digital computer according to claim **32** wherein each of said first and second processors include a central processing unit, a first memory storing program instructions and a second memory, separate and distinct from said first memory, storing data.

43. The digital computer according to claim **33** wherein at least one of said first memories is operable in a read-only mode of operation in which said program instructions are protected from alteration and erasure by a corresponding one of said central processing units.

44. A method of accessing a digital storage device using a digital computer system, the digital computer system including first and second independent local buses, first and second central processing units respectively connected to said first and second local buses, and a manually operative switch, the method comprising the steps of:

transmitting control signals from said first and second central processing units to respective ones of said fist and second local buses;

operating said switch to alternatively provide ones of said control signals from said first and second local buses to the digital storage device and to select a protected mode of operation thereof;

selectively operating the digital storage device in said protected mode of operation, said protected mode of operation including at least one of a write-only and read-only mode of operation; and

selectively operating said digital storage device responsive to said control signals in (i) a read mode of operation for reading previously stored data and (ii) a write mode of operation for storing data.

45. A method of accessing a digital storage device using a digital computer system, the digital computer system

including first and second system buses electrically independent of each other, master and slave central processing units connected to respective ones of said system buses, and a manually operative switch, the method comprising the steps of:

transmitting a write control signal from one of said master and salve processing units;

selectively storing data on said data storage device responsive to said write control signal; and

operating said switch to selectively enable and disable receipt by the data storage device of said write control signal from said first and second system buses.

* * * * *

US006035374A

United States Patent [19]

Panwar et al.

[11] Patent Number: 6,035,374

[45] Date of Patent: *Mar. 7, 2000

[54] **METHOD OF EXECUTING CODED INSTRUCTIONS IN A MULTIPROCESSOR HAVING SHARED EXECUTION RESOURCES INCLUDING ACTIVE, NAP, AND SLEEP STATES IN ACCORDANCE WITH CACHE MISS LATENCY**

[75] Inventors: **Ramesh Panwar**; **Joseph I. Chamdani**, both of Santa Clara, Calif.

[73] Assignee: **Sun Microsystems, Inc.**, Palo Alto, Calif.

[*] Notice: This patent is subject to a terminal disclaimer.

[21] Appl. No.: **08/881,239**

[22] Filed: **Jun. 25, 1997**

[51] **Int. Cl.**[7] **G06F 9/30**

[52] **U.S. Cl.** **711/118**; 711/118; 711/130; 711/148; 711/169; 712/10; 712/16; 712/32

[58] **Field of Search** 395/800.15, 500, 395/707, 708, 709, 710, 371, 182.13

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,287,508	2/1994	Hejna, Jr.	709/102
5,361,337	11/1994	Okin	395/569
5,487,156	1/1996	Popescu et al.	712/217
5,561,778	10/1996	Popescu et al.	712/239
5,625,837	4/1997	Popescu et al.	712/23
5,761,515	6/1998	Barton, III	395/709

OTHER PUBLICATIONS

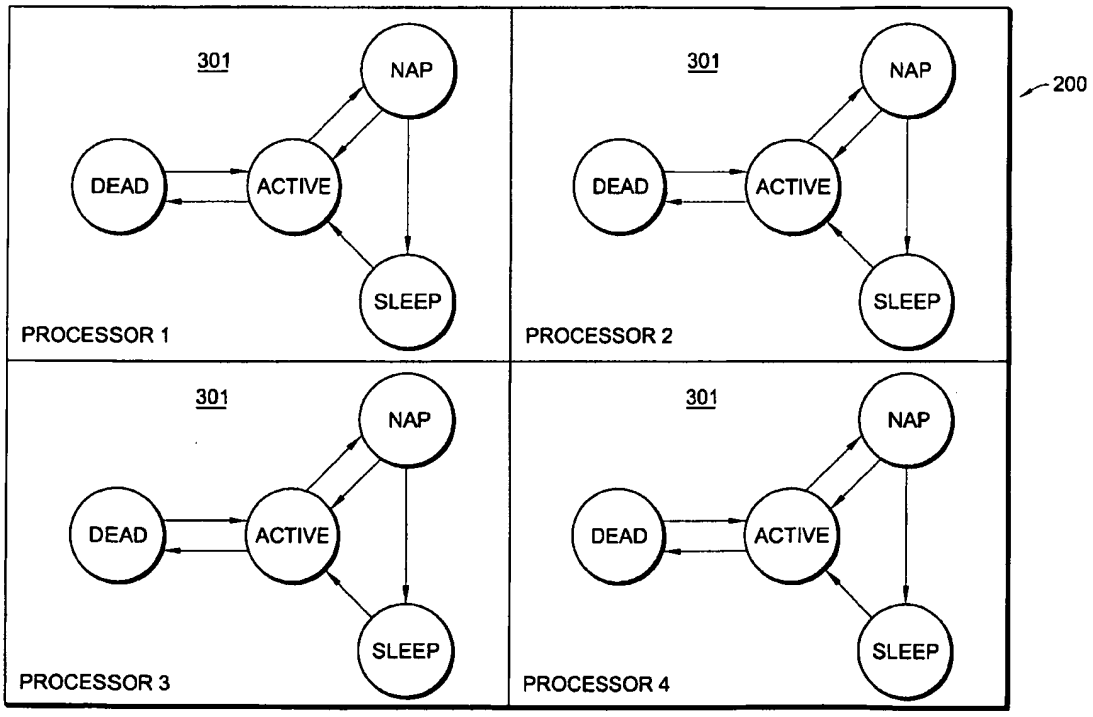
Greenley et al, "UltraSPARC: the next generation superscalar 64-bit SPARC", cOMPCON '95 'Technologies for the information Superhighway', Digest of Papers., pp. 442–449, Mar. 5, 1995.

Primary Examiner—Meng-At T. An
Assistant Examiner—Stacy Whitmore
Attorney, Agent, or Firm—William J. Kubida; Stuart T. Langley; Hogan & Hartson LLP

[57] **ABSTRACT**

A method of executing coded instructions in a dynamically configurable multiprocessor having shared execution resources including steps of placing a first processor in an active state upon booting of the multiprocessor. In response to a processor create command, a second processor is placed in an active state. When either the first or second processor encounter a cache miss that has to be serviced by off-chip cache the processor requiring service is placed in nap state in which instruction fetching for that processor is disabled. When either the first or second processor encounter a cache miss that has to be serviced by main memory, the processor requiring services I placed in a sleep state by flushing all instructions from the processor in the sleep state and disabling instruction fetching for the processor in the sleep state.

12 Claims, 10 Drawing Sheets



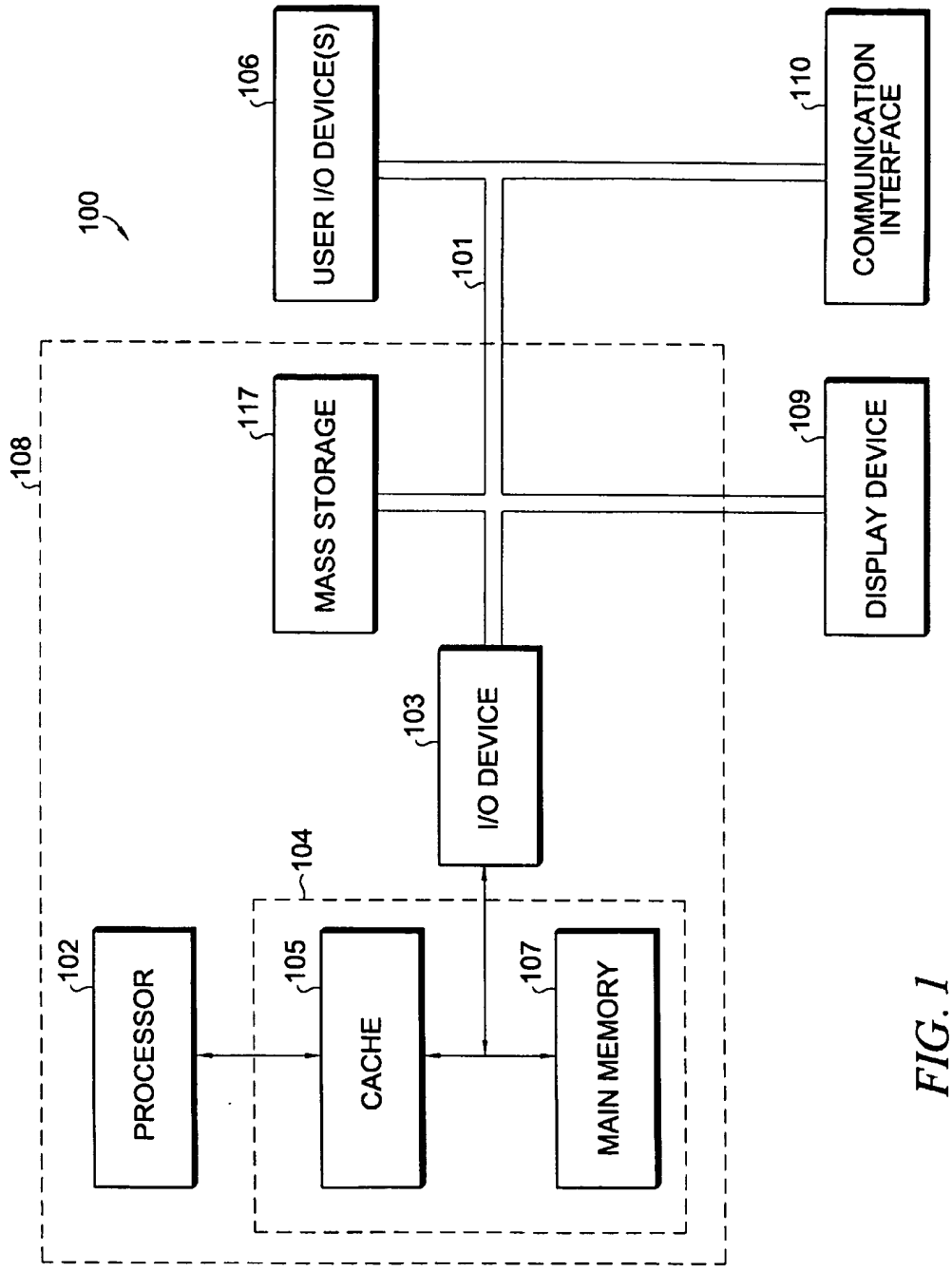

100
101
102
PROCESSOR
103
I/O DEVICE
104
105
CACHE
106
USER I/O DEVICE(S)
107
MAIN MEMORY
108
109
DISPLAY DEVICE
110
COMMUNICATION INTERFACE
117
MASS STORAGE

FIG. 1

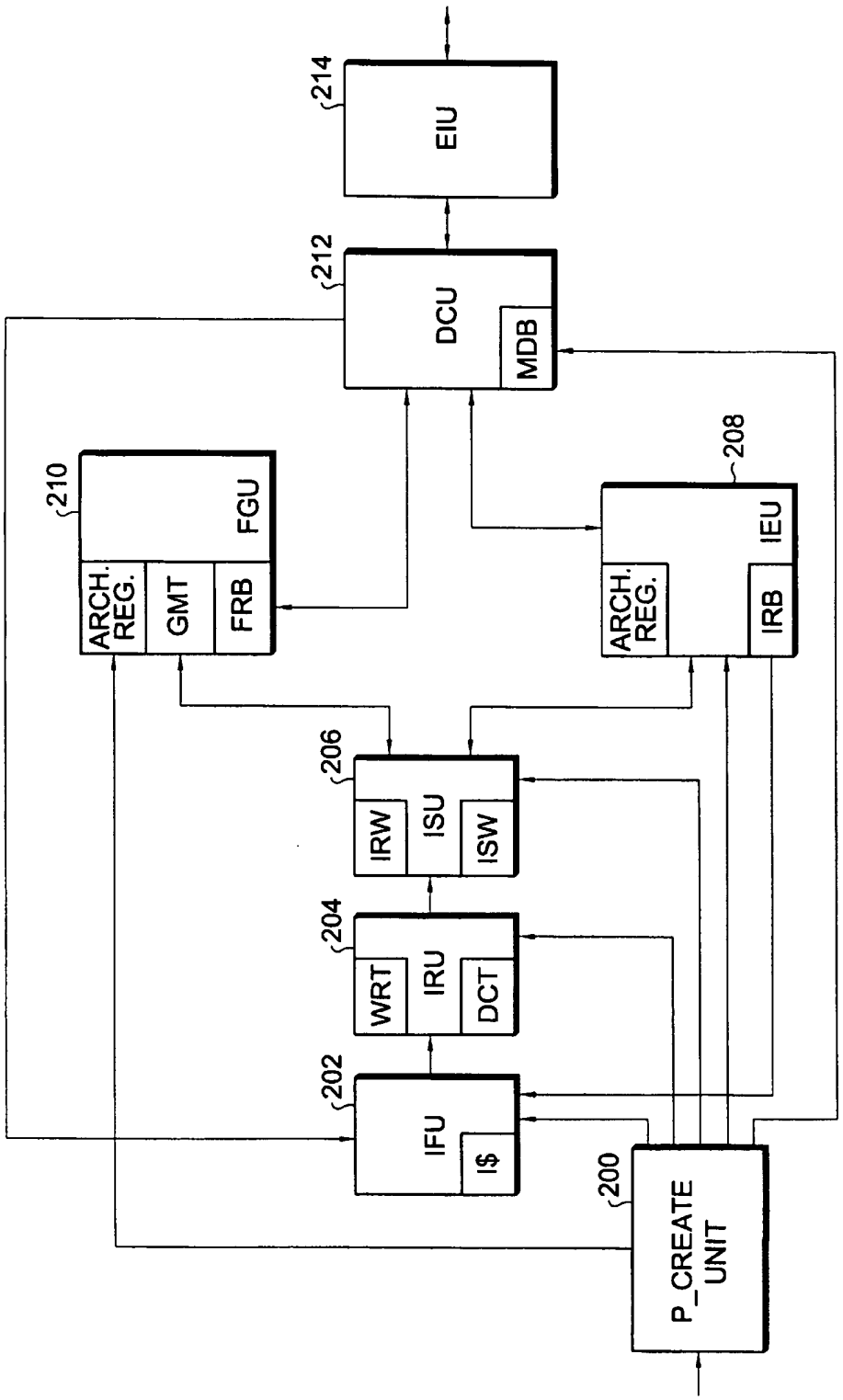

214
EIU
212
DCU
MDB
210
FGU
ARCH. REG.
GMT
FRB
208
IEU
ARCH. REG.
IRB
206
IRW
ISU
ISW
204
WRT
IRU
DCT
202
IFU
I$
200
P_CREATE UNIT

FIG. 2

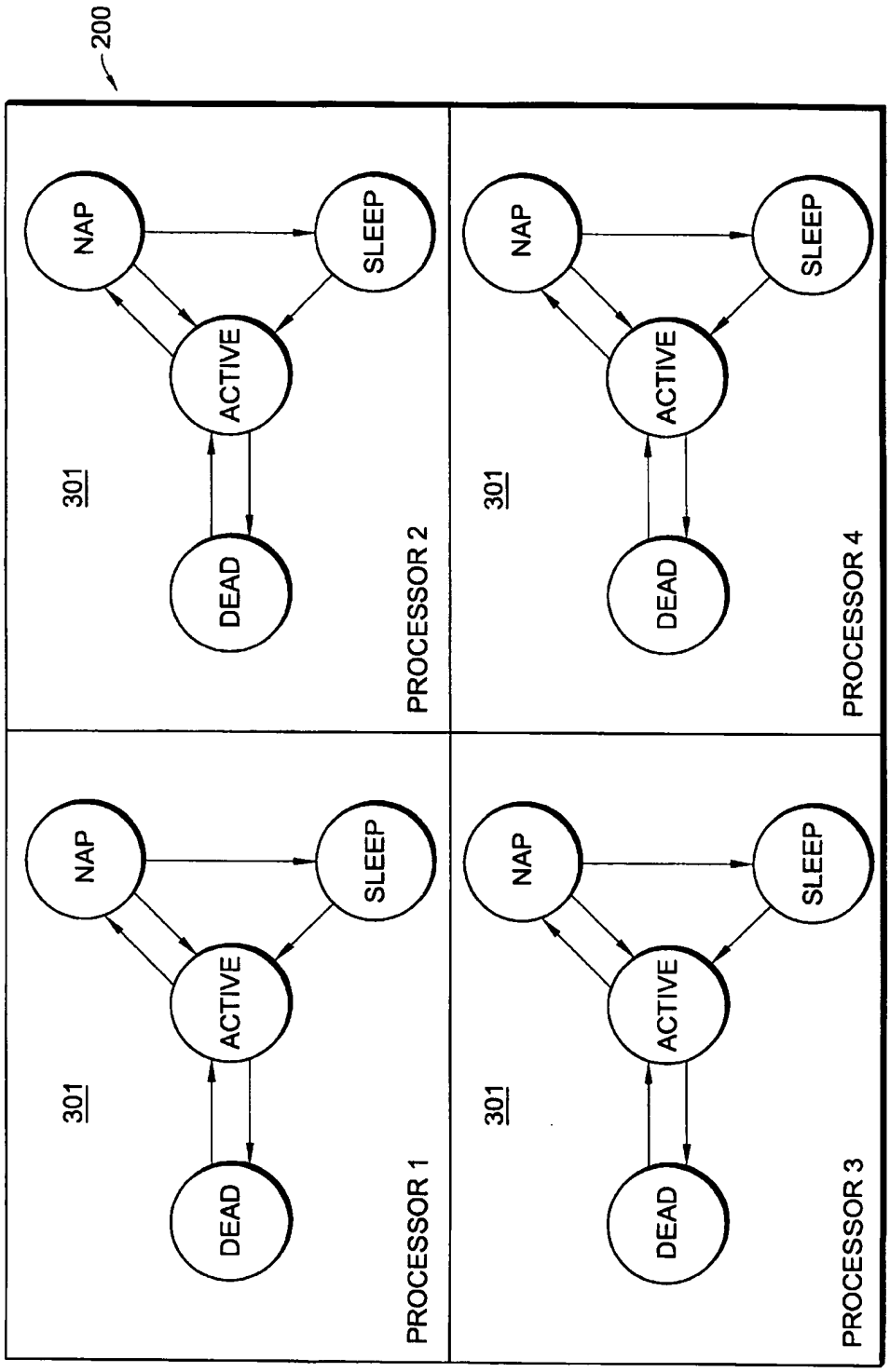

200
NAP
SLEEP
ACTIVE
DEAD
301
PROCESSOR 1
PROCESSOR 2
PROCESSOR 3
PROCESSOR 4

FIG. 3

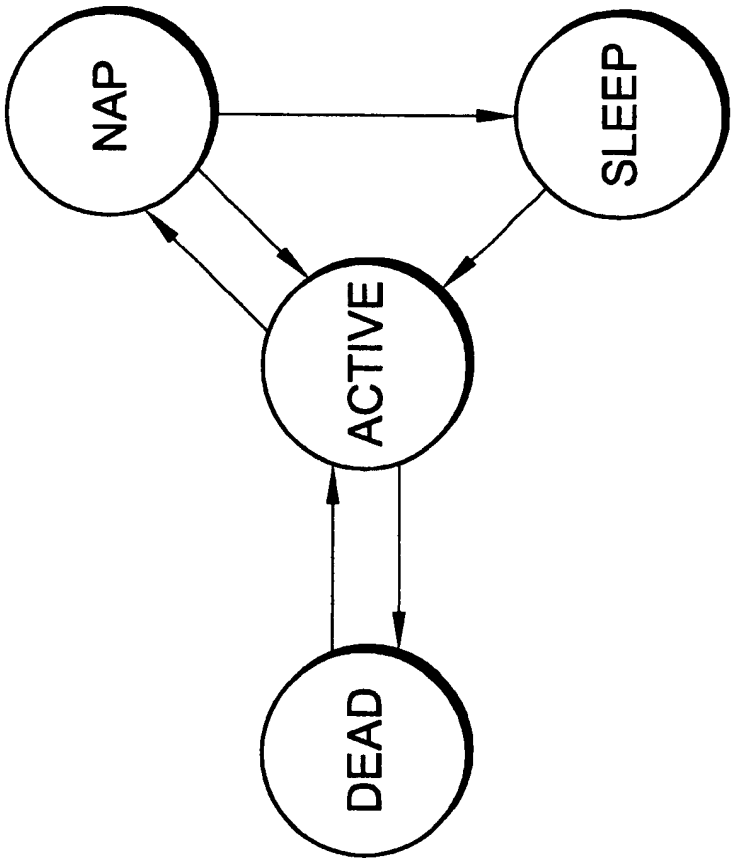


FIG. 4

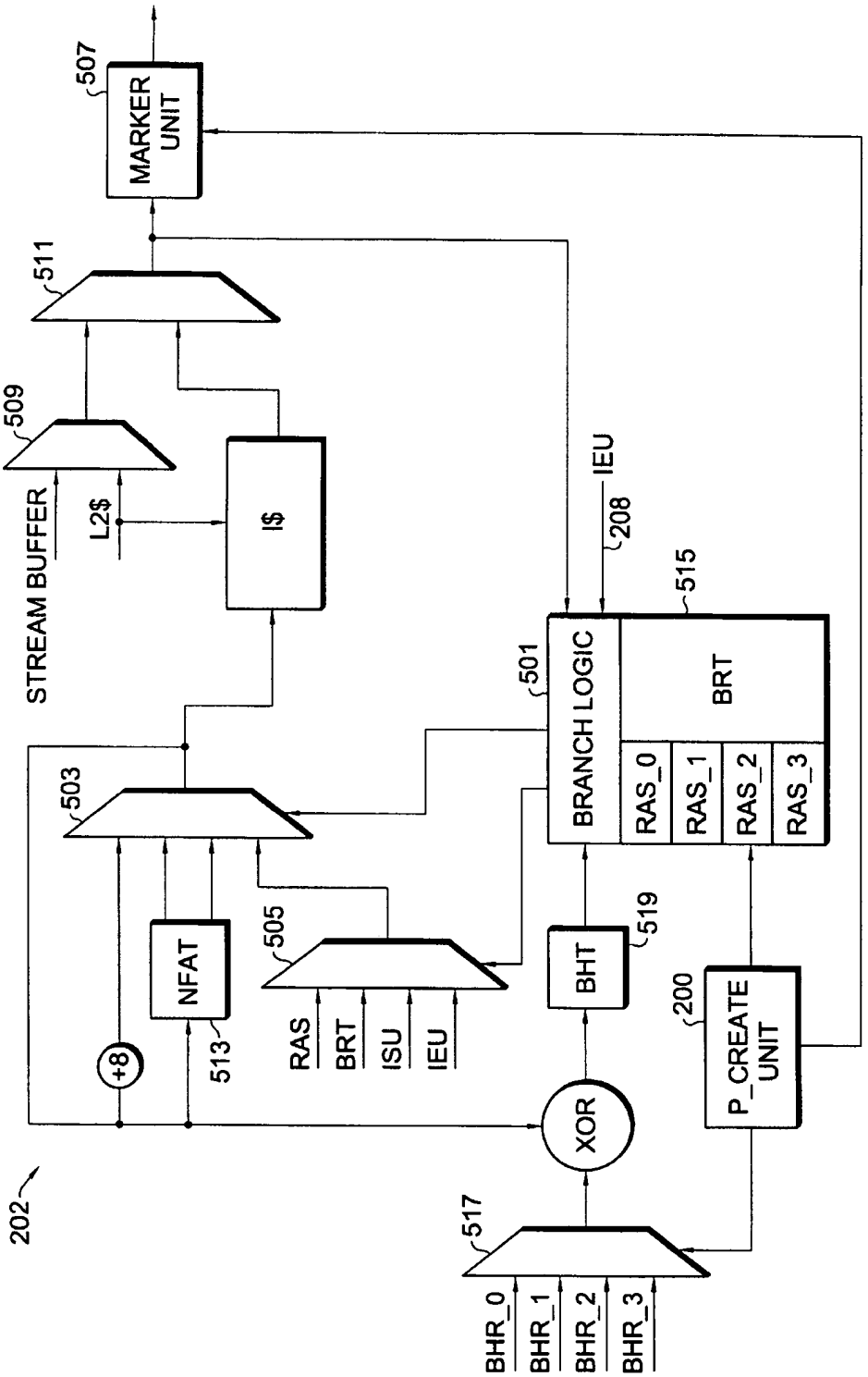

507
MARKER UNIT
511
509
STREAM BUFFER
L2$
I$
IEU
208
515
501
BRANCH LOGIC
BRT
RAS_0
RAS_1
RAS_2
RAS_3
503
NFAT
505
519
BHT
200
P_CREATE UNIT
+8
513
RAS
BRT
ISU
IEU
XOR
202
517
BHR_0
BHR_1
BHR_2
BHR_3

FIG. 5

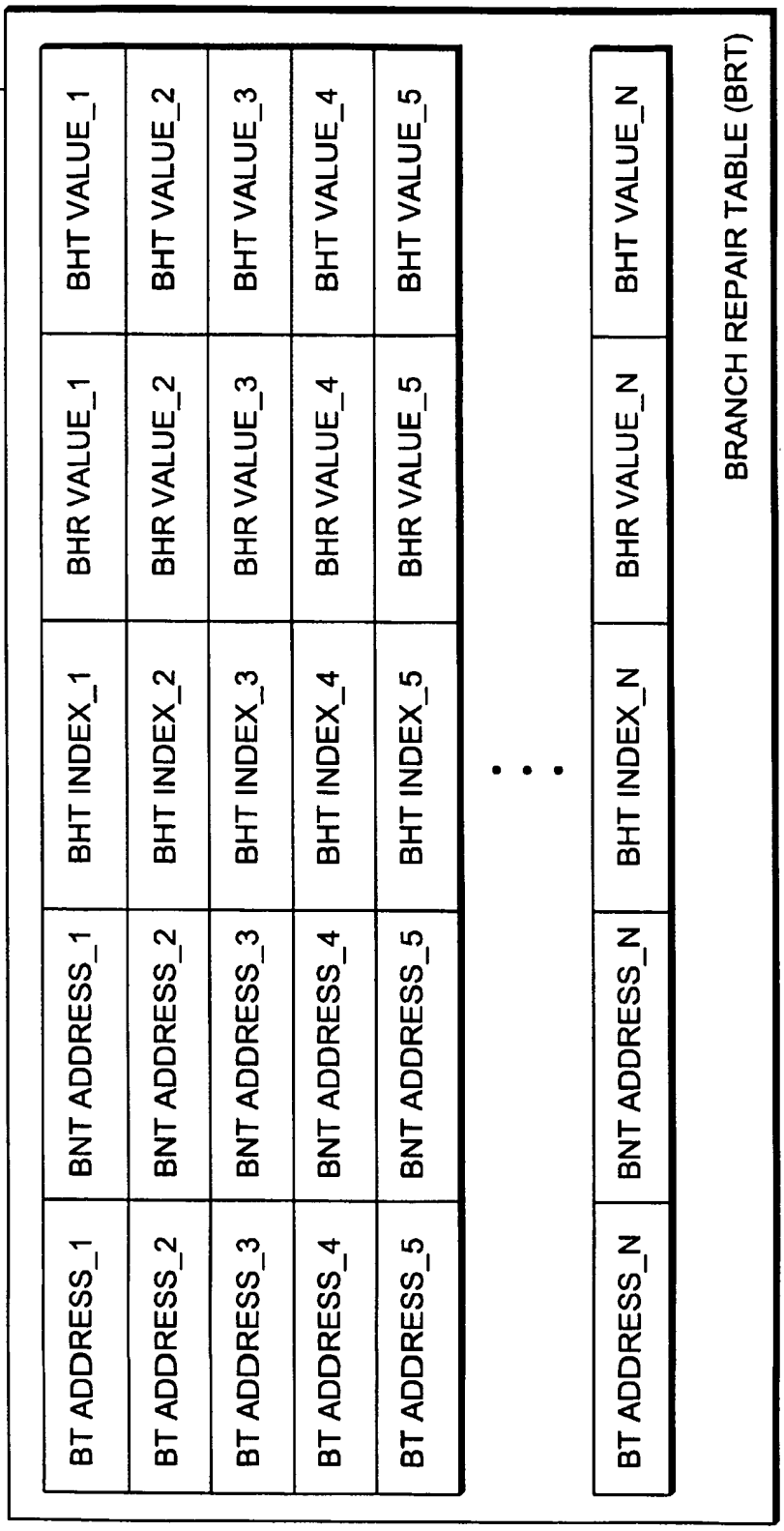

515
BT ADDRESS_1
BNT ADDRESS_1
BHT INDEX_1
BHR VALUE_1
BHT VALUE_1
BT ADDRESS_2
BNT ADDRESS_2
BHT INDEX_2
BHR VALUE_2
BHT VALUE_2
BT ADDRESS_3
BNT ADDRESS_3
BHT INDEX_3
BHR VALUE_3
BHT VALUE_3
BT ADDRESS_4
BNT ADDRESS_4
BHT INDEX_4
BHR VALUE_4
BHT VALUE_4
BT ADDRESS_5
BNT ADDRESS_5
BHT INDEX_5
BHR VALUE_5
BHT VALUE_5
BT ADDRESS_N
BNT ADDRESS_N
BHT INDEX_N
BHR VALUE_N
BHT VALUE_N
BRANCH REPAIR TABLE (BRT)

FIG. 6

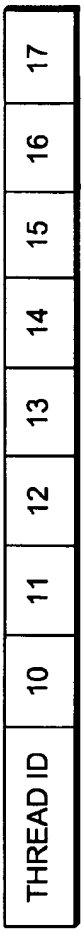

700
THREAD ID
10
11
12
13
14
15
16
17

FIG. 7

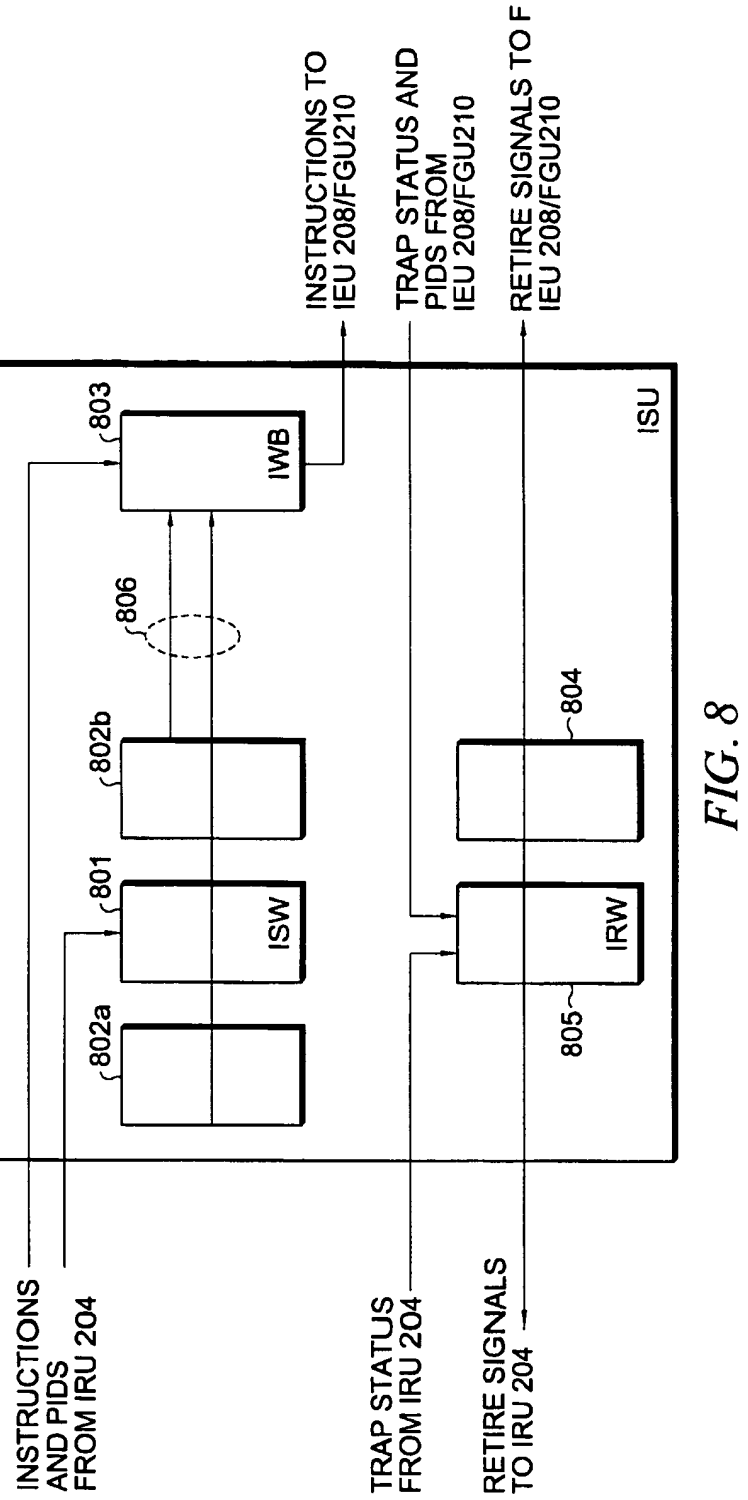

INSTRUCTIONS TO IEU 208/FGU210
TRAP STATUS AND PIDS FROM IEU 208/FGU210
RETIRE SIGNALS TO F IEU 208/FGU210
206
803
IWB
806
802b
801
ISW
802a
804
IRW
805
ISU
INSTRUCTIONS AND PIDS FROM IRU 204
TRAP STATUS FROM IRU 204
RETIRE SIGNALS TO IRU 204

FIG. 8

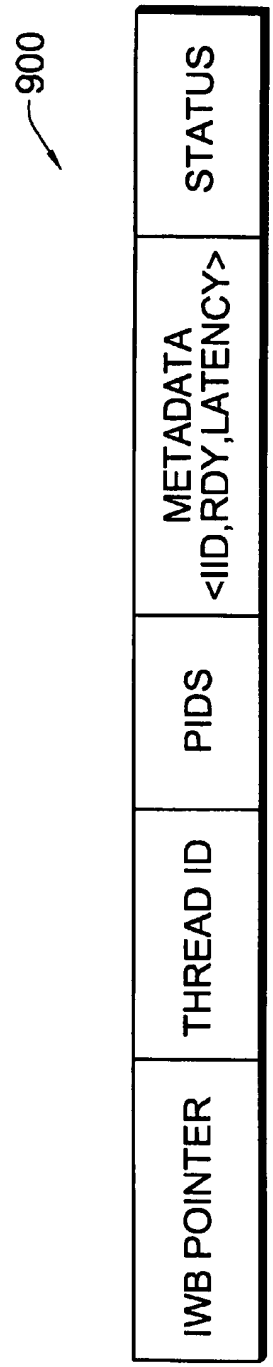

900
IWB POINTER
THREAD ID
PIDS
METADATA
<IID,RDY,LATENCY>
STATUS

FIG. 9

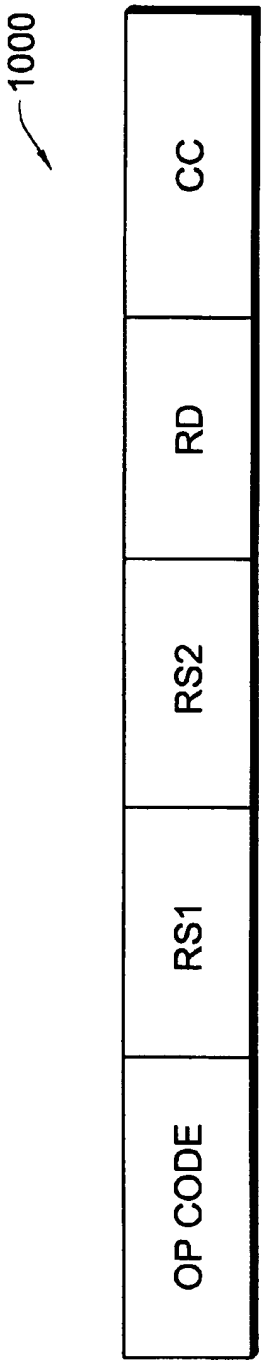

1000
OP CODE
RS1
RS2
RD
CC

FIG. 10

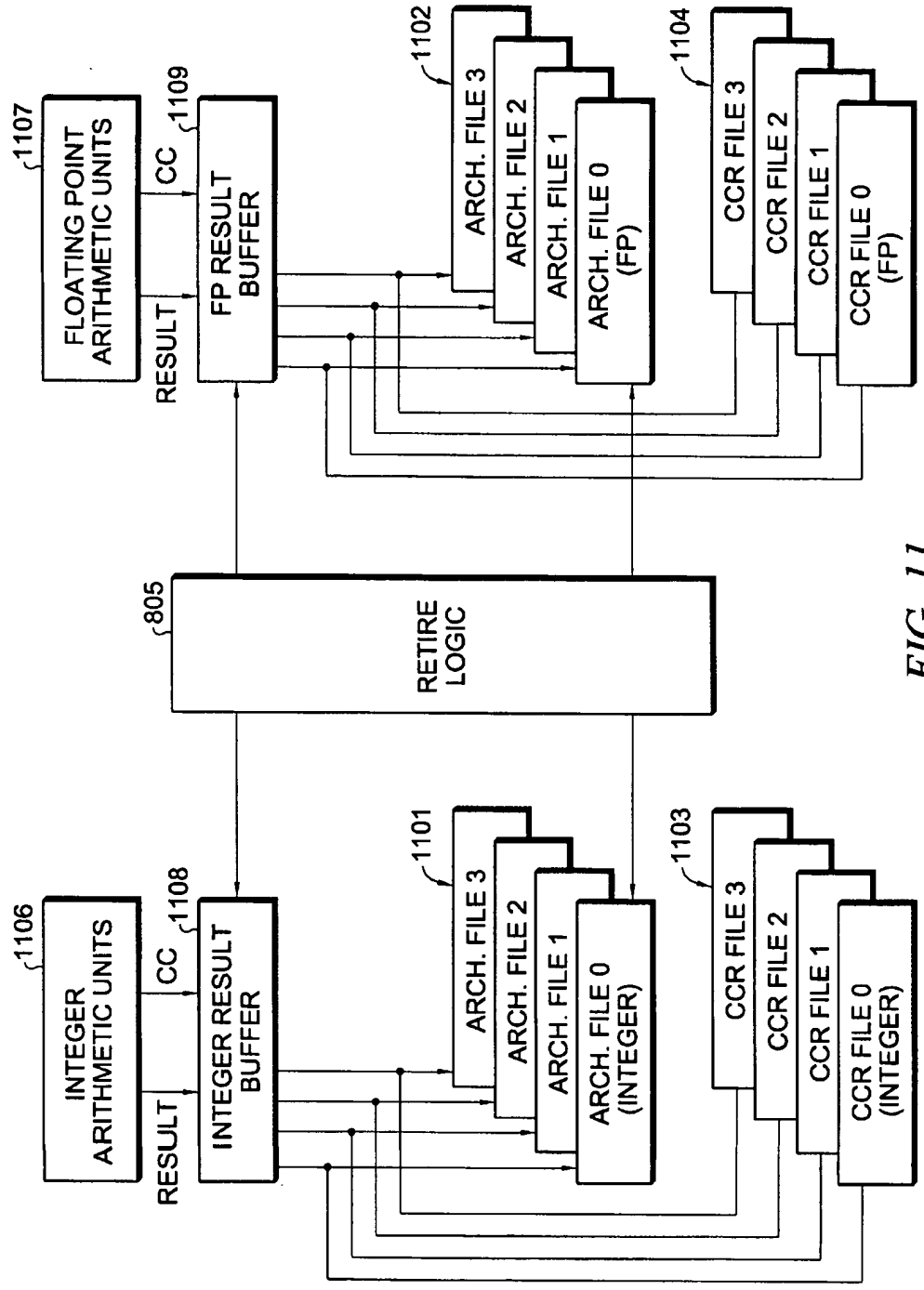

1107
FLOATING POINT ARITHMETIC UNITS
CC
RESULT
1109
FP RESULT BUFFER
1102
ARCH. FILE 3
ARCH. FILE 2
ARCH. FILE 1
ARCH. FILE 0 (FP)
1104
CCR FILE 3
CCR FILE 2
CCR FILE 1
CCR FILE 0 (FP)
805
RETIRE LOGIC
1106
INTEGER ARITHMETIC UNITS
CC
RESULT
1108
INTEGER RESULT BUFFER
1101
ARCH. FILE 3
ARCH. FILE 2
ARCH. FILE 1
ARCH. FILE 0 (INTEGER)
1103
CCR FILE 3
CCR FILE 2
CCR FILE 1
CCR FILE 0 (INTEGER)

FIG. 11

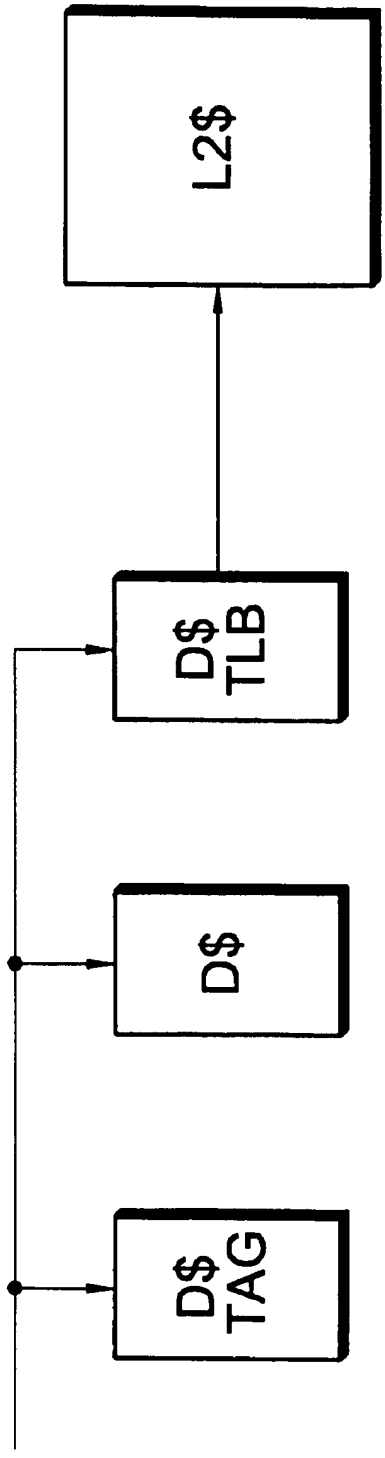


FIG. 12

METHOD OF EXECUTING CODED INSTRUCTIONS IN A MULTIPROCESSOR HAVING SHARED EXECUTION RESOURCES INCLUDING ACTIVE, NAP, AND SLEEP STATES IN ACCORDANCE WITH CACHE MISS LATENCY

CROSS-REFERENCES TO RELATED APPLICATIONS

The subject matter of the present application is related to that of co-pending U.S. patent application Ser. No. 08/881,958 identified as Docket No. P2345/37178.830071.000 for AN APPARATUS FOR HANDLING ALIASED FLOATING-POINT REGISTERS IN AN OUT-OF-ORDER PROCESSOR filed concurrently herewith by Ramesh Panwar; Ser. No. 08/881,729 identified as Docket No. P2346/37178.830072.000 for APPARATUS FOR PRECISE ARCHITECTURAL UPDATE IN AN OUT-OF-ORDER PROCESSOR filed concurrently herewith by Ramesh Panwar and Arjun Prabhu; Ser. No. 08/881,726 identified as Docket No. P2348/37178.830073.000 for AN APPARATUS FOR NON-INTRUSIVE CACHE FILLS AND HANDLING OF LOAD MISSES filed concurrently herewith by Ramesh Panwar and Ricky C. Hetherington; Ser. No. 08/881,908 identified as Docket No. P2349/37178.8300074.000 for AN APPARATUS FOR HANDLING COMPLEX INSTRUCTIONS IN AN OUT-OF-ORDER PROCESSOR filed concurrently herewith by Ramesh Panwar and Dani Y. Dakhil; Ser. No. 08/882,173 identified as Docket No. P2350/37178.830075.000 for AN APPARATUS FOR ENFORCING TRUE DEPENDENCIES IN AN OUT-OF-ORDER PROCESSOR filed concurrently herewith by Ramesh Panwar and Dani Y. Dakhil; Ser. No. 08/881,145 identified as Docket No. P2351/37178.830076.000 for APPARATUS FOR DYNAMICALLY RECONFIGURING A PROCESSOR filed concurrently herewith by Ramesh Panwar and Ricky C. Hetherington; Ser. No. 08/881,732 identified as Docket No. P2353/37178.830077.000 for APPARATUS FOR ENSURING FAIRNESS OF SHARED EXECUTION RESOURCES AMONGST MULTIPLE PROCESSES EXECUTING ON A SINGLE PROCESSOR filed concurrently herewith by Ramesh Panwar and Joseph I. Chamdani; Ser. No. 08/882,175 identified as Docket No. P2355/37178.830078.000 for SYSTEM FOR EFFICIENT IMPLEMENTATION OF MULTI-PORTED LOGIC FIFO STRUCTURES IN A PROCESSOR filed concurrently herewith by Ramesh Panwar; Ser. No. 08/882,311 identified as Docket No. P2365/37178.830080.000 for AN APPARATUS FOR MAINTAINING PROGRAM CORRECTNESS WHILE ALLOWING LOADS TO BE BOOSTED PAST STORES IN AN OUT-OF-ORDER MACHINE filed concurrently herewith by Ramesh Panwar, P. K. Chidambaran and Ricky C. Hetherington; Ser. No. 08/881,731 identified as Docket No. P2369/37178.830081.000 for APPARATUS FOR TRACKING PIPELINE RESOURCES IN A SUPERSCALAR PROCESSOR filed concurrently herewith by Ramesh Panwar; Ser. No. 08/882,525 identified as Docket No. P2370/37178.830082.000 for AN APPARATUS FOR RESTRAINING OVEREAGER LOAD BOOSTING IN AN OUT-OF-ORDER MACHINE filed concurrently herewith by Ramesh Panwar and Ricky C. Hetherington; Ser. No. 08/882,220 identified as Docket No. P2371/37178.830083.000 for AN APPARATUS FOR HANDLING REGISTER WINDOWS IN AN OUT-OF-ORDER PROCESSOR filed concurrently herewith by Ramesh Panwar and Dani Y. Dakhil; Ser. No. 08/881,847 identified as Docket No. P2372/37178.830084.000 for AN APPARATUS FOR DELIVERING PRECISE TRAPS AND INTERRUPTS IN AN OUT-OF-ORDER PROCESSOR filed concurrently herewith by Ramesh Panwar; Ser. No. 08/881,728 identified as Docket No. P2398/37178.830085.000 for NON-BLOCKING HIERARCHICAL CACHE THROTTLE filed concurrently herewith by Ricky C. Hetherington and Thomas M. Wicki; Ser. No. 08/881,727 identified as Docket No. P2406/37178.830086.000 for NON-THRASHABLE NON-BLOCKING HIERARCHICAL CACHE filed concurrently herewith by Ricky C. Hetherington, Sharad Mehrotra and Ramesh Panwar; Ser. No. 08/881,065 identified as Docket No. P2408/37178.830087.000 for IN-LINE BANK CONFLICT DETECTION AND RESOLUTION IN A MULTI-PORTED NON-BLOCKING CACHE filed concurrently herewith by Ricky C. Hetherington, Sharad Mehrotra and Ramesh Panwar; and Ser. No. 08/882,613 identified as Docket No. P2434/37178.830088.000 for SYSTEM FOR THERMAL OVERLOAD DETECTION AND PREVENTION FOR AN INTEGRATED CIRCUIT PROCESSOR filed concurrently herewith by Ricky C. Hetherington and Ramesh Panwar, the disclosures of which applications are herein incorporated by this reference. identified as Docket No. P2406/37178.830086.000 for NON-THRASHABLE NON-BLOCKING HIERARCHICAL CACHE filed concurrently herewith by Ricky C. Hetherington, Sharad Mehrotra and Ramesh Panwar; Ser. No. 08/881,065 identified as Docket No. P2408/37178.830087.000 for IN-LINE BANK CONFLICT DETECTION AND RESOLUTION IN A MULTI-PORTED NON-BLOCKING CACHE filed concurrently herewith by Ricky C. Hetherington, Sharad Mehrotra and Ramesh Panwar; and Ser. No. 08/882,613 identified as Docket No. P2434/37178.830088.000 for SYSTEM FOR THERMAL OVERLOAD DETECTION AND PREVENTION FOR AN INTEGRATED CIRCUIT PROCESSOR filed concurrently herewith by Ricky C. Hetherington and Ramesh Panwar, the disclosures of which applications are herein incorporated by this reference.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates in general to microprocessors and, more particularly, to a system, method, and processor architecture for dynamically reconfiguring a processor between uniprocessor and selected multiprocessor configurations.

2. Relevant Background

Early computer processors (also called microprocessors) included a central processing unit or instruction execution unit that executed only one instruction at a time. As used herein the term processor includes complete instruction set computers (CISC), educed instruction set computers (RISC) and hybrids. The processor executes programs having instructions stored in main memory by fetching their instruction, decoding them, and executing them one after the other. In response to the need for improved performance several techniques have been used to extend the capabilities of these early processors including pipelining, superpipelining, superscaling, speculative instruction execution, and out-of-order instruction execution.

Pipelined architectures break the execution of instructions into a number of stages where each stage corresponds to one step in the execution of the instruction. Pipelined designs increase the rate at which instructions can be executed by allowing a new instruction to begin execution before a

previous instruction is finished executing. Pipelined architectures have been extended to "superpipelined" or "extended pipeline" architectures where each execution pipeline is broken down into even smaller stages (i.e., microinstruction granularity is increased). Superpipelining increases the number of instructions that can be executed in the pipeline at any given time.

"Superscalar" processors generally refer to a class of microprocessor architectures that include multiple pipelines that process instructions in parallel. Superscalar processors typically execute more than one instruction per clock cycle, on average. Superscalar processors allow parallel instruction execution in two or more instruction execution pipelines. The number of instructions that may be processed is increased due to parallel execution. Each of the execution pipelines may have differing number of stages. Some of the pipelines may be optimized for specialized functions such as integer operations or floating point operations, and in some cases execution pipelines are optimized for processing graphic, multimedia, or complex math instructions.

The goal of superscalar and superpipeline processors, is to execute multiple instructions per cycle (IPC). Instruction-level parallelism (ILP) available in programs written to operate on the processor can be exploited to realize this goal. However, many programs are not coded in a manner that can take full advantage of deep, wide instruction execution pipelines in modern processors. Many factors such as low cache hit percentage, instruction interdependency, frequent access to slow peripherals, and the like cause the resources of a superscalar processor to be used inefficiently.

Superscalar architectures require that instructions be dispatched for execution at a sufficient rate. Conditional branching instructions create a problem for instruction fetching because the instruction fetch unit (IFU) cannot know with certainty which instructions to fetch until the conditional branch instruction is resolved. Also, when a branch is detected, the target address of the instructions following the branch must be predicted to supply those instructions for execution.

Recent processor architectures use a branch prediction unit to predict the outcome of branch instructions allowing the fetch unit to fetch subsequent instructions according to the predicted outcome. These instructions are "speculatively executed" to allow the processor to make forward progress during the time the branch instruction is resolved.

Another solution to increased processing power is provided by multiprocessing. Multiprocessing is a hardware and operating system feature that allows multiple processors to work together to share workload within a computing system. In a shared memory multiprocessing system, all processors have access to the same physical memory. One limitation of multiprocessing is that programs that have not been optimized to run as multiple process may not realize significant performance gain from multiple processors. However, improved performance is achieved where the operating system is able to run multiple programs concurrently, each running on a separate processor.

Multithreaded software is a recent development that allows applications to be split into multiple independent threads such that each thread can be assigned to a separate processor and executed independently parallel as if it were a separate program. The results of these separate threads are reassembled to produce a final result. By implementing each thread on a separate processor, multiple tasks are handled in a fast, efficient manner. The use of multiple processors allows various tasks or functions to be handled by other than a single CPU so that the computer power of the overall system is enhanced. However, because conventional multiprocessors are implemented using a plurality of discrete integrated circuits, communication between the devices limits its system clock frequency and the ability to share resources amongst the plurality of processors. As a result, conventional multiprocessor architectures result in duplication of resources which increases cost and complexity.

Given the wide variety and mix of software used on general purpose processors, it often occurs that some programs run most efficiently on superscalar, superpipeline uniprocessors while other programs run most efficiently in a multiprocessor environment. Moreover, the more efficient architecture may change over time depending on the mix of programs running at any given time. Because the architecture was defined by the CPU manufacturer and system board producer, end users and programmers had little or no ability to configure the architecture to most efficiently use the hardware resources to accomplish a given set of tasks.

SUMMARY OF THE INVENTION

Briefly stated, the present invention involves a method of executing coded instructions in a dynamically configurable multiprocessor having shared execution resources including steps of placing a first processor in an active state upon booting of the multiprocessor. In response to a processor create command, a second processor is placed in an active state. When either the first or second processor encounter a cache miss that has to be serviced by off-chip cache the processor requiring service is placed in nap state in which instruction fetching for that processor is disabled. When either the first or second processor encounter a cache miss that has to be serviced by main memory, the processor requiring services I placed in a sleep state by flushing all instructions from the processor in the sleep state and disabling instruction fetching for the processor in the sleep state.

A processor in accordance with the present invention includes a processor creation unit responsive to a processor create command to output signals indicating a current processor configuration and plurality of virtual or logical processors each virtual processor having a first set of execution resources that are uniquely identified with the virtual processor and a second set of execution resources that are shared amongst the plurality of virtual processors. A plurality of state machines responsive to the processor creation unit are provided, each corresponding to a selected one of the plurality of virtual processors. The state machines maintain processor status information representative of whether the processor is available to receive and execute instructions. The processor further includes status logic analyzing expected latency of instructions on each processor and updating the state machine corresponding to any processor having an instruction with an expected latency greater than a preselected threshold.

The foregoing and other features, utilities and advantages of the invention will be apparent from the following more particular description of a preferred embodiment of the invention as illustrated in the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 shows in block diagram form a computer system incorporating an apparatus and system in accordance with the present invention;

FIG. 2 shows a processor in block diagram form incorporating the apparatus and method in accordance with the present invention;

FIG. 3 illustrates a processor create unit in accordance with the present invention;

FIG. 4 shows a portion of the processor create unit of FIG. 3 in greater detail;

FIG. 5 shows an instruction fetch unit in accordance with the present invention in block diagram form;

FIG. 6 illustrates an example format for a branch repair table used in the fetch unit of FIG. 3;

FIG. 7 illustrates an example instruction bundle in accordance with an embodiment of the present invention;

FIG. 8 shows in block diagram form an instruction scheduling unit shown in FIG. 2;

FIG. 9 shows an exemplary entry in an instruction scheduling window in accordance with the present invention;

FIG. 10 shows an exemplary instruction wait buffer used in conjunction with the instruction scheduling window shown in FIG. 9; and

FIG. 11 shows in block diagram form instruction execution units in accordance with an embodiment of the present invention.

FIG. 12 shows in block diagram form a data cache (D$), data cache tag (D$TAG), and data cache translation lookaside buffer (D$TLB) connected to an L2 cache (L2$).

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention recognizes the wide variation in software (i.e., computer instruction code) that must be accommodated by a general purpose processor. Some code is most efficiently executed on a single high-speed processor with multiple deep pipelines. However, some applications cannot take advantage of these processor architectures.

Also, older software that was written before superscalar processors were common may not be optimized to take advantage of the benefits of multiple pipeline execution. Further, many applications now use multithreading software techniques that are best implemented on a multiprocessor platform rather than a single processor platform. The method, processor, and computer system in accordance with the present invention allows the processor hardware to be dynamically configured to meet the needs of a particular software application.

In such a dynamically configurable multiprocessor, however, many execution resources may be shared among the multiple virtual or logical processors on a single integrated circuit chip. Fairness issues may arise between the processes if, for example, one process misses in the cache more frequently than the others. In this case, the process that misses in the cache occupies space in the shared resources without doing any useful work while the cache miss is serviced by higher cache levels or main memory. The present invention recognizes this fairness issue with a solution for allocating resources fairly amongst the active processes.

Computer systems and processor architectures can be represented as a collection of interacting functional units as shown in FIG. 1 and FIG. 2. These functional units, discussed in greater detail below, perform the functions of storing instruction code, fetching instructions and data from memory, preprocessing fetched instructions, scheduling instructions to be executed, executing the instructions, managing memory transactions, and interfacing with external circuitry and devices.

The present invention is described in terms of apparatus and methods particularly useful in a superpipelined and superscalar processor **102** shown in block diagram form in FIG. 1 and FIG. 2. The particular examples represent implementations useful in high clock frequency operation and processors that issue and executing multiple instructions per cycle (IPC). However, it is expressly understood that the inventive features of the present invention may be usefully embodied in a number of alternative processor architectures that will benefit from the performance features of the present invention. Accordingly, these alternative embodiments are equivalent to the particular embodiments shown and described herein.

FIG. 1 shows a typical general purpose computer system **100** incorporating a processor **102** in accordance with the present invention. Computer system **100** in accordance with the present invention comprises an address/data bus **101** for communicating information, processor **102** coupled with bus **101** through input/output (I/O) device **103** for processing data and executing instructions, and memory system **104** coupled with bus **101** for storing information and instructions for processor **102**. Memory system **104** comprises, for example, cache memory **105** and main memory **107**. Cache memory **105** includes one or more levels of cache memory. In a typical embodiment, processor **102**, I/O device **103**, and some or all of cache memory **105** may be integrated in a single integrated circuit, although the specific components and integration density are a matter of design choice selected to meet the needs of a particular application.

User I/O devices **106** are coupled to bus **101** and are operative to communicate information in appropriately structured form to and from the other parts of computer **100**. User I/O devices may include a keyboard, mouse, card reader, magnetic or paper tape, magnetic disk, optical disk, or other available input devices, include another computer. Mass storage device **117** is coupled to bus **101** and is implemented using one or more magnetic hard disks, magnetic tapes, CDROMs, large banks of random access memory, or the like. A wide variety of random access and read only memory technologies are available and are equivalent for purposes of the present invention. Mass storage **117** may include computer programs and data stored therein. Some or all of mass storage **117** may be configured to be incorporated as a part of memory system **104**.

In a typical computer system **100**, processor **102**, I/O device **103**, memory system **104**, and mass storage device **117**, are coupled to bus **101** formed on a printed circuit board and integrated into a single housing as suggested by the dashed-line box **108**. However, the particular components chosen to be integrated into a single housing is based upon market and design choices. Accordingly, it is expressly understood that fewer or more devices may be incorporated within the housing suggested by dashed line **108**.

Display device **109** is used to display messages, data, a graphical or command line user interface, or other communications with the user. Display device **109** may be implemented, for example, by a cathode ray tube (CRT) monitor, liquid crystal display (LCD) or any available equivalent.

FIG. 2 illustrates principle components of processor **102** in greater detail in block diagram form. It is contemplated that processor **102** may be implemented with more or fewer functional units and still benefit from the apparatus and methods of the present invention unless expressly specified herein. Also, functional units are identified using a precise nomenclature for ease of description and understanding, but other nomenclature often is often used by various manufacturers to identify equivalent functional units.

Unlike conventional multiprocessor architectures, the present invention may be, and desirably is, implemented as a single circuit on a single integrated circuit chip. In this manner, the individual processors are not only closely coupled, but are in essence merged such that they can share resources efficiently amongst the processors. This resources sharing simplifies many of the communication overhead problems inherent in other multiprocessor designs. For example, memory, including all levels of the cache subsystem, are easily shared among the processor and so cache coherency is not an issue. Although the resources are shared, the multiprocessor configuration in accordance with the present invention achieves the same advantages as conventional multiprocessing architectures by enabling independent threads and processes to execute independently and in parallel.

In accordance with the present invention, processor create unit **200** is coupled to receive a processor create instruction from either the computer operating system, a running application, or through a hardware control line (not shown). In a specific example, the processor create instruction is added to the SPARC V9 instruction architecture as a privileged command that can be issued only by the operating system. The processor create instruction instructs processor **102** to reconfigure as either a uniprocessor or as one of an available number of multiprocessor configurations by specifying a number of virtual processors. In a specific example, one virtual processor is created for each thread or process in the instruction code. In this manner, when it is determined by the operating system, application, or otherwise that the current instruction code can be executed more efficiently in a multiprocessor of n-processors, the processor create instruction is used to instantiate n virtual processors to execute the code. The configuration may change dynamically in response to new applications starting or a running application spawning a new thread.

The term "virtual processors" is used herein to describe the functional operation of the dynamically configurable processor and method in accordance with the present invention. Each virtual processor is a logical processor as opposed to a physically implemented processor. As described in greater detail below, each virtual processor requires a set of execution resources that are unique to that processor. These unique resources are enabled in response to the processor create command to activate a virtual processor. Also, each virtual processor requires access to a set of shared execution resources. These shared resources are enabled independently of the processor create command. In accordance with the present invention, when a virtual processor is activated, the processor behaves as if it is the selected uniprocessor or multiprocessor, however, no physical reconfiguration, rewiring, or the like is required.

Referring to FIG. **3** and FIG. **4**, processor creation unit **200** may be implemented as a plurality of state machines **301**. In the example of FIG. **3**, one state machine **301** is provided for each virtual processor. Any number of state machines **301**, hence any number of virtual processors, may be included in processor **102**. One of the state machines **301** is designated as a primary unit that is analogous to a boot processor in a conventional multiprocessor design. The primary state machine **301** will become active automatically when processor **102** is activated, while the other state machines **301** wait to respond to the processor create command to become activated.

At a minimum, each state machine comprises a "dead" or inactive state and a "live" or active state. The transition between dead and active states is controlled by the processor create command. Optionally, a processor destroy command can also be provided to move a state machine **301** from an active state to a dead state. Desirably, each state machine **301** includes a "nap" state that can be reached from the active state, and a "sleep" state that can be reached from the nap state. The active state can be reached from the dead, nap, or sleep states as shown in FIG. **4**.

In a particular implementation, a virtual processor in the active state is assigned exclusive control over some of the shared resources in the functional units of processor **102**. When one of the virtual processors experiences a delay in executing instructions, that delay preferably does not affect the other virtual processors. For example, when one virtual processor experiences an on-chip cache miss, it will require tens of clock cycles to obtain the required data from the off-chip cache. When an off-chip cache miss occurs and data must be retrieved from main memory, or mass storage, hundreds of clock cycles may occur before that process can make forward progress.

The nap and sleep states in state machines **301** are provided to account for these delays. When a virtual processor encounters an on-chip cache miss it is placed in a nap state. The nap state disables instruction fetching only for the virtual processor in the nap state. Instruction fetching continues for the remaining virtual processors. In the nap state, instruction scheduling and execution remain enabled (described in greater detail hereinbelow). Hence, in the nap state a virtual processor is allowed to continue possession of execution resources that it has already occupied, but is not allowed to take possession of any more resources so that other virtual processors may use these resources.

When a napping virtual processor encounters a cache miss that must be satisfied by main memory, or mass storage, the virtual processor enters the sleep state. In the sleep state, all instructions belonging to the sleeping virtual processor are flushed from ISU **206**. Hence, not only is the sleeping processor prevented from taking additional resources, but it is also forced to release resources previously occupied so that other virtual processors may continue execution unimpaired. The sleep state prevents instructions from the sleeping virtual processor from clogging up ISU **206** and thereby interfering with execution of instructions from other virtual processors.

Instruction fetch unit (IFU) **202** (shown in greater detail in FIG. **5**) comprises instruction fetch mechanisms and includes, among other things, an instruction cache I$ for storing instructions, branch prediction logic **501**, and address logic for addressing selected instructions in instruction cache I$. The instruction cache I$ is a portion of the level one (L1) cache with another portion (D$, not shown) of the L1 cache dedicated to data storage in a Harvard architecture cache. Other cache organizations are known, including unified cache structures, and may be equivalently substituted and such substitutions will result in predictable performance impact.

IFU **202** fetches one or more instructions each clock cycle by appropriately addressing the instruction cache I$ via MUX **503** and MUX **305** under control of branch logic **501** as shown in FIG. **5**. In the absence of a conditional branch instruction, IFU **202** addresses the instruction cache sequentially. Fetched instructions are passed to IRU **204** shown in FIG. **2**. Any fetch bundle may include multiple control-flow (i.e., conditional or unconditional branch) instructions. Hence, IFU **202** desirably bases the next fetch address decision upon the simultaneously predicted outcomes of multiple branch instructions.

The branch prediction logic **501** (shown in FIG. **5**) handles branch instructions, including unconditional branches. An outcome for each branch instruction is predicted using any of a variety of available branch prediction algorithms and mechanisms. In the example of FIG. **5**, an exclusive-OR operation is performed on the current address and a value from a selected branch history register (BHR) to generate an index to the branch history table (BHT) **519**. To implement a multiprocessor in accordance with the present invention, each virtual processor has a unique BHR. For a four processor implementation shown in FIG. **5**, four BHR inputs labeled BHR_**0**, BHR_**1**, BHR_**2**, and BHR_**3** are provided.

Each active BHR comprises information about the outcomes of a preselected number of most-recently executed condition and unconditional branch instructions for a particular active virtual processor. For virtual processors in the dead state, the BHR value is a don't care. An outcome can be represented in binary as taken or not taken. In a specific example, each active BHR comprises a seventeen-bit value representing the outcomes of seventeen most-recently executed branch instructions.

Processor create unit **200** selects one active BHR using multiplexor **517**. Only one BHR is selected at a time, and processor create unit **200** serves to select the BHR in a round-robin fashion each clock cycle from the virtual processors that are in an active state. Hence, if only one processor is active, only BHR_**0** will be selected. Each BHR comprises the outcomes (i.e., taken or not taken) for a number of most-recently executed conditional and unconditional branch instructions occurring on a processor-by-processor basis. In a specific example, each BHR comprises a 17-bit value. When a conditional branch instruction is predicted, the predicted outcome is used to speculatively update the appropriate BHR so that the outcome will be a part of the information used by the next BHT access for that virtual processor. When a branch is mispredicted, however, the appropriate BHR must be repaired by transferring the BHR VALUE from BRT **515**, along actual outcome of the mispredicted branch are loaded into the BHR corresponding to the virtual processor on which the branch instruction occurred.

Next fetch address table (NFAT) **513** determines the next fetch address based upon the current fetch address received from the output of MUX **503**. For example, NFAT **513** may comprise 2048 entries, each of which comprises two multi-bit values corresponding to a predicted next fetch address for instructions in two halves of the current fetch bundle. Two bits of the multi-bit values comprise set prediction for the next fetch, while the remaining bits are used to index the instruction cache IS and provide a cache line offset in a specific implementation.

A branch repair table (BRT) **515** comprises entries or slots for a number of unresolved branch instructions. BRT **515** determines when a branch is mispredicted based upon input from IEU **208**, for example. BRT **515**, operating through branch logic **501**, redirects IFU **202** down the correct branch path. Each entry in BRT **515** comprises multiple fields as detailed in FIG. **6**. Branch taken fields (i.e., BT ADDRESS_**1** through BT ADDRESS_N) store an address (i.e., program counter value) for the first fetch bundle in the branch instructions predicted path. Branch not taken fields (i.e., BNT ADDRESS_**1** through BNT ADDRESS_N) store an address for the first fetch bundle in a path not taken by the branch instruction. A branch history table (BHT) index (BHT INDEX_**1**-BHT INDEX_N) points to a location in the branch history table that was used to predict the branch instruction. The BHR VALUE and BHT VALUE fields store the value of the BHR and BHT, respectively, at the time a branch instruction was predicted.

The branch history table (BHT) **519** comprises a plurality of two-bit values. More than two-bit values may be used, but acceptable results are achieved with two bits. BHT **519** is indexed by computing an exclusive-or of the selected BHR value with the current fetch address taken from the output of MUX **503**. In a specific example, the 17 least significant bits of the current address are used in the XOR computation (excluding the two most-least significant bits which are always 0's in a byte addressed processor with 32-bit instructions) to match the 17 bit values in each BHR. The XOR computation generates a 17-bit index that selects one entry in BHT. The 17 bit index enables selection from up to 2^{17} or 128K locations in BHT **519**. One BHT **519** may be shared among any number of virtual processors.

Once a branch is resolved, the address of the path this branch actually follows is communicated from IEU **208** and compared against the predicted path address store in the BT ADDRESS fields. If these two addresses differ, those instructions down the mispredicted path are flushed from the processor and IFU **202** redirects instruction fetch down the correct path identified in the BNT ADDRESS field using the BRT input to MUX **505**. Once a branch is resolved, the BHT value is updated using the BHT index and BHT value stored in BRT **515**. In the example of FIG. **5**, each entry in BHT **519** is a two-bit saturating counter. When a predicted branch is resolved taken, the entry used to predict this outcome is incremented. When a predicted branch is resolved not taken, the entry in BHT **519** is decremented. Other branch prediction algorithms and techniques may be used in accordance with the present invention, so long as care is taken to duplicate resources on a processor-by-processor basis where those resources are used exclusively by a given processor.

Although the fields in BRT **515** may include a thread identifier field to indicate which virtual processor executed the branch instruction assigned to that slot, BRT **515** is shared among all of the virtual processors and requires little modification to support dynamically configurable uniprocessing and multiprocessing in accordance with the present invention.

Another resource in IFU **202** that must be duplicated for each virtual processor is the return address stack (RAS) labeled RAS_**0** through RAS_**3** in FIG. **5**. Each RAS comprises a last in, first out (LIFO) stack in a particular example that stores the return addresses of a number of most-recently executed branch and link instructions. These instructions imply a subsequent RETURN instruction that will redirect processing back to a point just after the fetch address when the branch or link instruction occurred. When an instruction implying a subsequent RETURN (e.g., a CALL or JMPL instruction in the SPARC V9 architecture) is executed, the current program counter is pushed onto a selected one of RAS_**0** through RAS_**3**. The RAS must be maintained on a processor-by-processor (i.e., thread-by-thread) basis to ensure return to the proper location.

When a subsequent RETURN instruction is executed, the program counter value on top of the RAS is popped and selected by appropriately controlling multiplexor **505** in FIG. **5**. This causes IFU **202** to begin fetching at the RAS-specified address. The RETURN instruction is allocated an entry in BRT **515** and the fall-through address is stored in the BNT ADDRESS field for that entry. If this RETURN instruction is mispredicted, it is extremely unlikely that the fall-through path is the path the RETURN

should follow and IFU **202** must be redirected via an address computed by IEU **208** and applied to the IEU input to multiplexor **505**.

IFU **202** includes instruction marker circuitry **507** for analyzing the fetched instructions to determine selected information about the instructions. Marker unit **507** is also coupled to processor create unit **200**. This selected information, including the thread identification (i.e., the virtual processor identification) generated by processor create unit **200**, is referred to herein as "instruction metadata". In accordance with the present invention, each fetch bundle is tagged with a thread identification for use by downstream functional units. Other metadata comprises information about, for example, instruction complexity and downstream resources that are required to execute the instruction. The term "execution resources" refers to architectural register space, rename register space, table space, decoding stage resources, and the like that must be committed within processor **102** to execute the instruction. The metadata can be generated by processor create unit **200** or dedicated combinatorial logic that outputs the metadata in response to the instruction op-code input. Alternatively, a look-up table or content addressable memory can be used to obtain the metadata. In a typical application, the instruction metadata will comprise two to eight bits of information that is associated with each instruction.

In many applications it is desirable to fetch multiple instructions at one time. For example, four, eight, or more instructions may be fetched simultaneously in a bundle. In accordance with the present invention, each instruction bundle includes the instruction metadata (e.g., THREAD ID) as shown in instruction bundle **700** shown in FIG. **7**. **I0–I7** represent conventional instruction fields that comprise, for example, an op-code, one or more operand or source register specifiers (typically denoted rs1, rs2, rs3, etc.) and a destination register specifier (typically denoted rd) and/or condition code specifiers. Other information, including instruction metadata, may be included in each **I0–I7** field. As shown in FIG. **7**, the instruction metadata for an entire bundle **700** may be grouped in a single field labeled THREAD ID in FIG. **7**. Alternatively, the instruction metadata may be distributed throughout the **I0–I7** instruction fields.

Although IFU **208** supporting dynamically configurable multiprocessing in accordance with the present invention has been described in terms of a specific processor capable of implementing one, two, three, or four virtual processors in a single processor unit, it should be appreciated that n-way multithreading can be achieved by modifying IFU **208** to fetch instructions from n different streams or threads on a round-robin or thread-by-thread basis each cycle. Because each fetch bundle includes instructions from only one thread, the modifications required to support dynamically configurable multithreading can be implemented with modest increase in hardware size and complexity. Essentially, any state information that needs to be tracked on a per-processor or per-thread basis must be duplicated. Other resources and information can be shared amongst the virtual processors. The BHR tracks branch outcomes within a single thread of execution so there should be one copy of the BHR for each thread. Similarly, the RAS tracks return addresses for a single thread of execution and so there should be one copy of the RAS for each thread.

The remaining functional units shown in FIG. **2** are referred to herein as "downstream" functional units although instructions and data flow bidirectionally between the remaining functional units. As described in greater detail below, some or all of the downstream functional units have resources that may be effectively shared among multiprocessors in accordance with the present invention. A significant advantage in accordance with the present invention is that the downstream functional units do not require complete duplication to enable multiprocessor functionality. Another advantage is that several functional units include resources that can be dynamically shared thereby enabling "on-the-fly" reconfiguration from a uniprocessor mode to any of a number of multiprocessor modes.

IRU **204**, comprises one or more pipeline stages that include instruction renaming and dependency checking mechanisms. A feature of the present invention is that inter-bundle dependency checking is relaxed because bundles from different threads are inherently independent. IRU **204** implements necessary logic for handling rename registers in a register window-type architecture such as the SPARC-V9 instruction architecture. A dependency checking mechanism, called an inverse map table (IMT) or dependency checking table (DCT) in a specific example, is used to analyze the instructions to determine if the operands (identified by the instructions' register specifiers) cannot be determined until another live instruction has completed. A particular embodiment of an IMT is described in greater detail in U.S. patent application Ser. No. 08/882,173 titled "A STRUCTION AND MECHANISM FOR ENFORCING TRUE DEPENDENCIES IN AN OUT OF ORDER MACHINE" by Ramesh Panwar and Dani Y. Dakhil filed concurrently herewith, is operative to map register specifiers in the instructions to physical register locations and to perform register renaming to prevent dependencies. IRU **204** outputs renamed instructions to instruction scheduling unit (ISU) **206**.

Each entry compares the source fields (rs1 and rs2) of all eight incoming instructions against the destination register field for that entry. If there is a match, the entry broadcasts its own address on to the corresponding bus through a simple encoder. This broadcast address is referred to as a producer ID (PID) and is used by the instruction scheduling window (ISW) within instruction scheduling unit **206** to determine the ready status of waiting instructions. A match also takes place between the CC fields of the eight incoming instructions and the CC field of the entry.

When a branch instruction is resolved and its predicted direction turns out to be wrong, the prefetched instructions following it (within the same thread or virtual processor) must be flushed from the ISW. Fetching into the window must resume at the position following the mispredicted branch, as described hereinbefore with respect to IFU **202**. However, instructions being flushed may have been taken over as being youngest producers of certain registers in the machine. There are two ways to handle this situation. One, resume fetching into the window but prevent scheduling of the new instructions until all of the previous instructions have retired from the window. Alternatively, rewind the youngest-producer information within the dependency checking table so the older instructions are reactivated as appropriate.

Each entry in the ISW is tagged with a two-bit thread ID to identify the thread to which the instruction belongs. On a flush, the ISW entries belonging to only the thread that suffered the branch mispredict are eliminated while the entries corresponding to the other threads stay resident. Hence, the flush information that is broadcast by IEU **208** has to contain the thread identifier of the mispredicted branch.

IRU **204** further comprises a window repair table (WRT) operative to store status information about register window

instructions used to restore the state of register windows after a branch misprediction. The WRT includes thirty-two entries or slots, each entry comprising one or more fields of information in a particular example. The number of entries in the WRT may be more or less depending on the needs of a particular application. The WRT can be shared amongst the multiprocessors in accordance with the present invention and does not require modification. The WRT would not be necessary in a processor that does not use register widows.

ISU **206** (shown in greater detail in FIG. **8**) is operative to schedule and dispatch instructions as soon as their dependencies have been satisfied into an appropriate execution unit (e.g., integer execution unit (IEU) **208**, or floating point and graphics unit (FGU) **210**). ISU **206** also maintains trap status of live instructions. ISU **206** may perform other functions such as maintaining the correct architectural state of processor **102**, including state maintenance when out-of-order instruction processing is used. ISU **206** may include mechanisms to redirect execution appropriately when traps or interrupts occur and to ensure efficient execution of multiple threads where multiple threaded operation is used. Multiple thread operation means that processor **102** is running multiple substantially independent processes simultaneously. Multiple thread operation is consistent with but not required by the present invention.

In accordance with an embodiment of the present invention, state machines **301** are implemented in ISU **206** by maintaining virtual processor status information in ISU **206**. Although other functional units use the thread ID to implement multiprocessors in accordance with the present invention, ISU **206** uses the virtual processor status information to implement the active, nap, and sleep states described hereinbefore. Hence, to ease circuit complexity and improve operation speed, it is advantageous to implement state machines **301** in ISU **206**.

ISU **206** also operates to retire executed instructions when completed by IEU **208** and FGU **210**. ISU assigns each live instruction a position or slot in an instruction retirement window (IRW) shown in FIG. **8**. In a specific embodiment, the IRW includes one slot for every live instruction. ISU **206** directs the appropriate updates to architectural register files and condition code registers upon complete execution of an instruction. ISU **206** is responsive to exception conditions and discards or flushes operations being performed on instructions subsequent to an instruction generating an exception. ISU **206** quickly removes instructions from a mispredicted branch and instructs IFU **202** to fetch from the correct branch. An instruction is retired when it has finished execution and all instructions from which it depends have completed. Upon retirement the instruction's result is written into the appropriate register file and is no longer deemed a "live instruction".

In operation, ISU **206** receives renamed instructions from IRU **204** and registers them for execution by assigning each instruction a position or slot in an instruction scheduling window (ISW). In a specific embodiment, the ISW includes one slot **900** (shown in FIG. **9**) for every live instruction. Each entry **900** in the ISW is associated with an entry **1000** in an instruction wait buffer (IWB) shown in FIG. **10** by an IWB POINTER. In accordance with the present invention, each entry **900** includes a THREAD ID field holding the thread identification. Dependency information about the instruction is encoded in the PID fields of ISW entry **900**. Metadata such as an instruction identification, ready status, and latency information, for example, are stored in a METADATA field of each entry **900**. Status information, included virtual processor status, is stored in the STATUS field ISW entry **900**. In a particular example, each STATUS field includes three bits to indicate status (e.g., active, dead, nap, sleep) for each instruction. State machines **301** are implemented by control logic that updates these status bits.

One or more instruction picker devices such as **802***a* and **802***b* in FIG. **8** pick instructions from the ISW that are ready for execution by generating appropriate signals on word lines **806** to the instruction wait buffer (IWB) so that the instruction will be read out or issued to execution units such as IEU **208** and FGU **210** in FIG. **2**. Pickers **802***a* and **802***b* desirably base the decision of which instructions to pick upon the instruction's relative age as well (e.g., how long the instruction has been in ISU **206**).

The instruction is issued to IEU **208** or FGU **210** together with the thread identification and instruction identification so that IEU **208** or FGU **210** can respond back with the trap and completion status on an instruction-by-instruction basis. When the trap and completion status of an instruction arrives from IEU **208** or FGU **210**, they are written into an instruction retirement window (IRW) shown in FIG. **2**. Retirement logic examines contiguous entries in the IRW and retires them in order to ensure proper architectural state update.

In addition to retirement, one or more instructions can be removed from the execution pipelines by pipeline flushes in response to branch mispredictions, traps, and the like. In the case of a pipeline flush, the resources committed to the flushed instructions are released as in the case of retirement, but any speculative results or state changes caused by the flushed instructions are not committed to architectural registers. In accordance with the present invention, a pipeline flush affects only instructions in a single thread or virtual processor, leaving other active virtual processors unaffected.

IEU **208** includes one or more pipelines, each pipeline comprising one or more stages that implement integer instructions such as integer arithmetic units **1106** in FIG. **11**. The integer arithmetic units **1106** are shared amongst the virtual processors in accordance with the present invention. IEU **208** also includes an integer result buffer (IRB) **1108** that is shared amongst the virtual processors for holding the results and state of speculatively executed integer instructions. IRB **1108** comprises a hardware-defined number of registers that represent another type of execution resource. In a specific example IRB **1108** comprises one register slot for each live instruction.

IEU **208** functions to perform final decoding of integer instructions before they are executed on the execution units and to determine operand bypassing amongst instructions in an out-of-order processor. IEU **208** executes all integer instructions including determining correct virtual addresses for load/store instructions. IEU **208** also maintains correct architectural register state for a plurality of architectural integer registers in processor **102**. IEU **208** preferably includes mechanisms to access single and/or double precision architectural registers **1101**. In accordance with the present invention, a copy of the integer architectural register files is provided for each virtual processor as shown in FIG. **11**. Similarly, a copy of the condition code architectural register files **1103** is provided for each virtual processor. Speculative results and condition codes in shared integer result buffer **1108** are transferred upon retirement to appropriate architectural files **1101** and **1103** under control of retire logic **804**. Because the architectural register files **1101** and **1103** may be much smaller than integer result buffer **1108**, duplication of the architectural files on a processor-by-processor basis has limited impact on the overall size and complexity of the dynamically reconfigurable multiprocessor in accordance with the present invention.

FGU **210**, includes one or more pipelines, each comprising one or more stages that implement floating point instructions such as floating point arithmetic units **1107** in FIG. **11**. FGU **210** also includes a floating point results buffer (FRB) **1109** for holding the results and state of speculatively executed floating point and graphic instructions. The FRB **1109** comprises a hardware-defined number of registers that represent another type of execution resource. In the specific example FRB **1109** comprises one register slot for each live instruction. FGU **210** functions to perform final decoding of floating point instructions before they are executed on the execution units and to determine operand bypassing amongst instructions in an out-of-order processor.

In a specific example, FGU **210** includes one or more pipelines (not shown) dedicated to implement special purpose multimedia and graphic instructions that are extensions to standard architectural instructions for a processor. FGU **210** may be equivalently substituted with a floating point unit (FPU) in designs in which special purpose graphic and multimedia instructions are not used. FGU **210** preferably includes mechanisms to access single and/or double precision architectural registers **1102** and condition code registers **1104**. Speculative results and condition codes in shared floating point result buffer **1109** are transferred upon retirement to appropriate architectural files **1102** and **1104** under control of retire logic **804**. Each processor is provided with a unique set of architectural registers **1102** and **1104** to provide processor independence.

Optionally, FGU **210** may include a graphics mapping table (GMT) comprising a fixed number of resources primarily or exclusively used for graphics operations. The GMT resources are typically used only for graphics instructions and so will not be committed for each live instruction. In accordance with the present invention, the instruction metadata includes information about whether the fetched instruction requires GMT-type resources. The GMT resources may be shared amongst the virtual processors in accordance with the present invention.

A data cache memory unit (DCU) **212**, including cache memory **105** shown in FIG. **1**, functions to cache memory reads from off-chip memory through external interface unit (EIU) **214** shown in FIG. **2**. Optionally, DCU **212** also caches memory write transactions. DCU **212** comprises one or more hierarchical levels of cache memory and the associated logic to control the cache memory. One or more of the cache levels within DCU **212** may be read only memory to eliminate the logic associated with cache writes.

In a specific implementation, DCU **212** comprises separate Instruction and Data caches in the L1 cache, a unified level 2 cache (L2$) that is desirably formed on-chip, and an external level 3 cache (L3$). Details on the size, organization and operational policy are discussed herein to ease description, but it is expressly understood that a wide variety of cache and memory architectures can cooperate with and benefit from the apparatus and methods in accordance with the present invention.

Each cache level has an inherently higher latency (i.e., time to return data). Latency is typically measured from the launch of the virtual address of a memory operation instruction. The first level caches (I$ and D$) have the lowest latency in the range of a few clock cycles. L2 cache is next with a latency of two to ten times that of the L1 cache. In the specific example L3 cache is an off-chip cache resulting in approximate latency of twenty-five to fifty clock cycles. In many designs, L2 cache is off-chip and so latency estimates would be adjusted accordingly. Latency to main memory is approximately 100 clock cycles although this number can vary dramatically if some of main memory is serviced by swap files in mass storage **107** (shown in FIG. **1**). For purposes of understanding the present invention, it is important only to know that each subsequent cache level results in increasing latency.

Each cache level includes some device to detect whether the data requested by a program or by a lower level of cache exists in the cache level. When data exists in the cache a "hit" is generated and the data is returned to service the memory operation instruction. When data does not exist in the cache, a "miss" is generated and the data must be fetched from a higher cache level or main memory. In accordance with the present invention, DCU **212** is coupled to update state machines **301** such that a cache miss in the on-chip cache(s) results in a transition from an active state to a nap state. The nap state prohibits instructions from being fetched for the napping processor. Further, a cache miss that must be serviced from main memory (including service from mass storage), places the processor generating the cache request into the sleep state. The sleep state results in termination of instruction fetching and instruction execution. Instruction execution is halted by, for example, flushing all instructions tagged with the thread id corresponding to the sleeping process from instruction scheduling unit **206**. By removing these instruction, pickers **802***a* and **802***b* can move forward to pick instructions from active processes.

It is contemplated that other instructions in addition to memory operations may result in latencies that can be handled by the state machine process in accordance with the present invention. For example, CISC machines may include instructions specifically adapted to access external peripherals, network resources, or the like. These instructions may also suffer from long expected latency and so are desirably placed in a nap or sleep state to avoid clogging shared resources in a dynamically configurable multiprocessor in accordance with the present invention.

While the invention has been particularly shown and described with reference to a preferred embodiment thereof, it will be understood by those skills in the art that various other changes in the form and details may be made without departing from the spirit and scope of the invention. The various embodiments have been described using hardware examples, but the present invention can be readily implemented in software. For example, it is contemplated that a programmable logic device, hardware emulator, software simulator, or the like of sufficient complexity could implement the present invention as a computer program product including a computer usable medium having computer readable code embodied therein for dynamically configuring emulated or simulated processor. Accordingly, these and other variations are equivalent to the specific implementations and embodiments described herein.

What is claimed is:

1. A method of executing coded instructions in a multiprocessor having shared execution resources comprising the steps of:

placing a first processor in an active state upon booting of the multiprocessor;

in response to a processor create command, placing a second processor in an active state while the first processor remains in the active state;

simultaneously executing instructions from each processor in an active state using the shared execution resources;

determining when either the first or second processor encounter a first cache miss that has to be serviced by off-chip cache;

in response to determining the first cache miss, placing the processor requiring service in nap state in which instruction fetching for that processor is disabled.

2. The method of claim 1 further comprising the steps of:

determining if the first cache miss is followed by a cache hit in the off-chip cache; and

in response to determining the off-chip cache hit, placing the processor requiring service in the active state.

3. The method of claim 1 further comprising steps of:

determining when either the first or second processor encounter a experience a second cache miss that has to be serviced by main memory; and

in response to determining the second cache miss, placing the processor requiring service in a sleep state in which instruction fetching and instruction execution for that processor are disabled.

4. The method of claim 3 wherein the step of placing the processor requiring service in a sleep state further comprises flushing all instructions from the processor in the sleep state and disabling instruction fetching for the processor in the sleep state.

5. The method of claim 3 further comprising the steps of:

determining when the second cache miss is serviced by the main memory; and

in response to determining that the second cache miss is serviced, placing the processor requiring service in the active state from the sleep state.

6. The method of claim 1 wherein the coded instructions comprise instructions from a number of threads and each thread is executed on a separate one of the processors such that placing one processor in the nap state does not affect instruction fetching and execution of any other processor.

7. The method of claim 3 further comprising the steps of:

detecting when the first and the second cache misses are serviced; and

placing the processor requiring service in an active state in response to the cache miss being serviced.

8. The method of claim 1 wherein the step of determining a first cache miss comprises comparing a tag portion of a physical address used as a target for a memory transaction to tag data for the cache location accessed by the target physical address.

9. A method of executing coded instructions in a multiprocessor having shared execution resources comprising the steps of:

dynamically allocating a first portion of the shared resources to a first processor;

dynamically allocating a second portion of the shared resources to a second processor;

placing the first processor in an active state;

placing the second processor in an active state while the first processor remains in the active state;

simultaneously executing instructions for both the first and second processor using the shared execution resources;

detecting when either the first or second processor encounters an instruction with a long expected latency;

in response to detecting the instruction, placing the processor associated with the detected instruction in nap state in which instruction fetching for that processor is disabled.

10. The method of claim 9 wherein the detected instruction involves a cache miss.

11. The method of claim 9 wherein the processor not associated with the detected instruction remains in the active state during the other processor's nap state.

12. The method of claim 9 wherein the step of fetching comprises:

identifying a thread to which the fetched instructions belong; and

tagging each fetched instruction with a thread identification that associates the fetched instruction with either the first or second processor.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,035,374
DATED : March 7, 2000
INVENTOR(S) : Panwar, et. al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 18, line 34, after instruction insert--"within the shared execution resources"--.

Signed and Sealed this

Sixth Day of February, 2001

Attest:

Attesting Officer

Q. TODD DICKINSON

Director of Patents and Trademarks

US006292874B1

(12) **United States Patent**
Barnett

(10) **Patent No.: US 6,292,874 B1**
(45) **Date of Patent: *Sep. 18, 2001**

(54) **MEMORY MANAGEMENT METHOD AND APPARATUS FOR PARTITIONING HOMOGENEOUS MEMORY AND RESTRICTING ACCESS OF INSTALLED APPLICATIONS TO PREDETERMINED MEMORY RANGES**

(75) Inventor: **Philip C. Barnett**, West End Witney (GB)

(73) Assignee: **Advanced Technology Materials, Inc.**, Danbury, CT (US)

(*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/420,318**

(22) Filed: **Oct. 19, 1999**

(51) **Int. Cl.**[7] **G06F 12/02**
(52) **U.S. Cl.** **711/153**; 711/173; 703/23
(58) **Field of Search** 711/115, 153, 711/163, 173; 713/200; 703/23, 24

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,930,129	*	5/1990	Takahira	714/766
5,206,938	*	4/1993	Fujioka	711/200
5,500,949	*	3/1996	Saito	711/100
5,517,014	*	5/1996	Iijima	235/492
5,890,199	*	3/1999	Downs	711/106
5,912,453	*	6/1999	Gungl et al.	235/492
5,912,849	*	6/1999	Yasu et al.	365/195

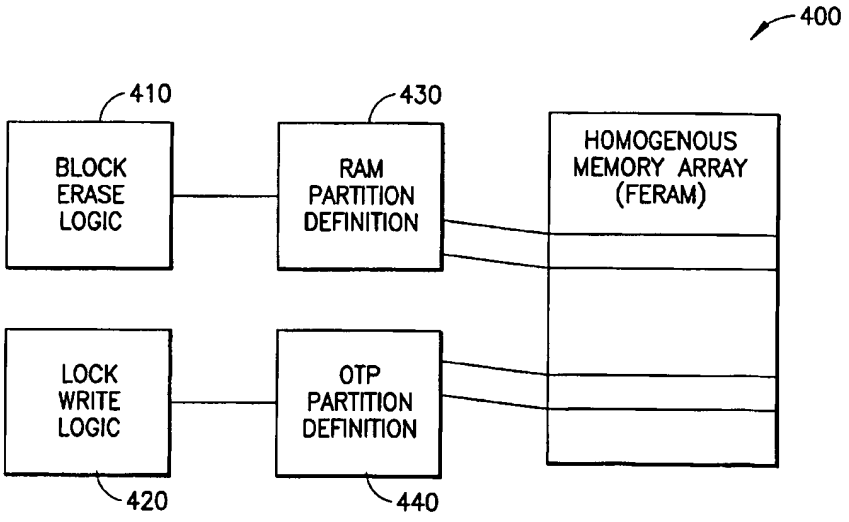
* cited by examiner

Primary Examiner—Hiep T. Nguyen
(74) *Attorney, Agent, or Firm*—Oliver A. Zitzmann; Kevin M. Mason

(57) **ABSTRACT**

A memory management unit is disclosed for a single-chip data processing circuit, such as a smart card. The memory management unit (i) partitions a homogeneous memory device to achieve heterogeneous memory characteristics for various regions of the memory device, and (ii) restricts access of installed applications executing in the microprocessor core to predetermined memory ranges. The memory management unit provides two operating modes for the processing circuit. In a secure kernel mode, the programmer can access all resources of the device including hardware control. In an application mode, the memory management unit translates the virtual memory address used by the software creator into the physical address allocated to the application by the operating system in a secure kernel mode during installation. The memory management unit implements memory address checking using limit registers and translates virtual addresses to an absolute memory address using offset registers. The memory management unit loads limit and offset registers with the appropriate values from an application table to ensure that the executing application only accesses the designated memory locations. The memory management unit can also partition a homogeneous memory device, such as an FERAM memory device, to achieve heterogeneous memory characteristics normally associated with a plurality of memory technologies, such as volatile, non-volatile and program storage (ROM) memory segments. Once partitioned, the memory management unit enforces the appropriate corresponding memory characteristics for each heterogeneous memory type.

18 Claims, 4 Drawing Sheets



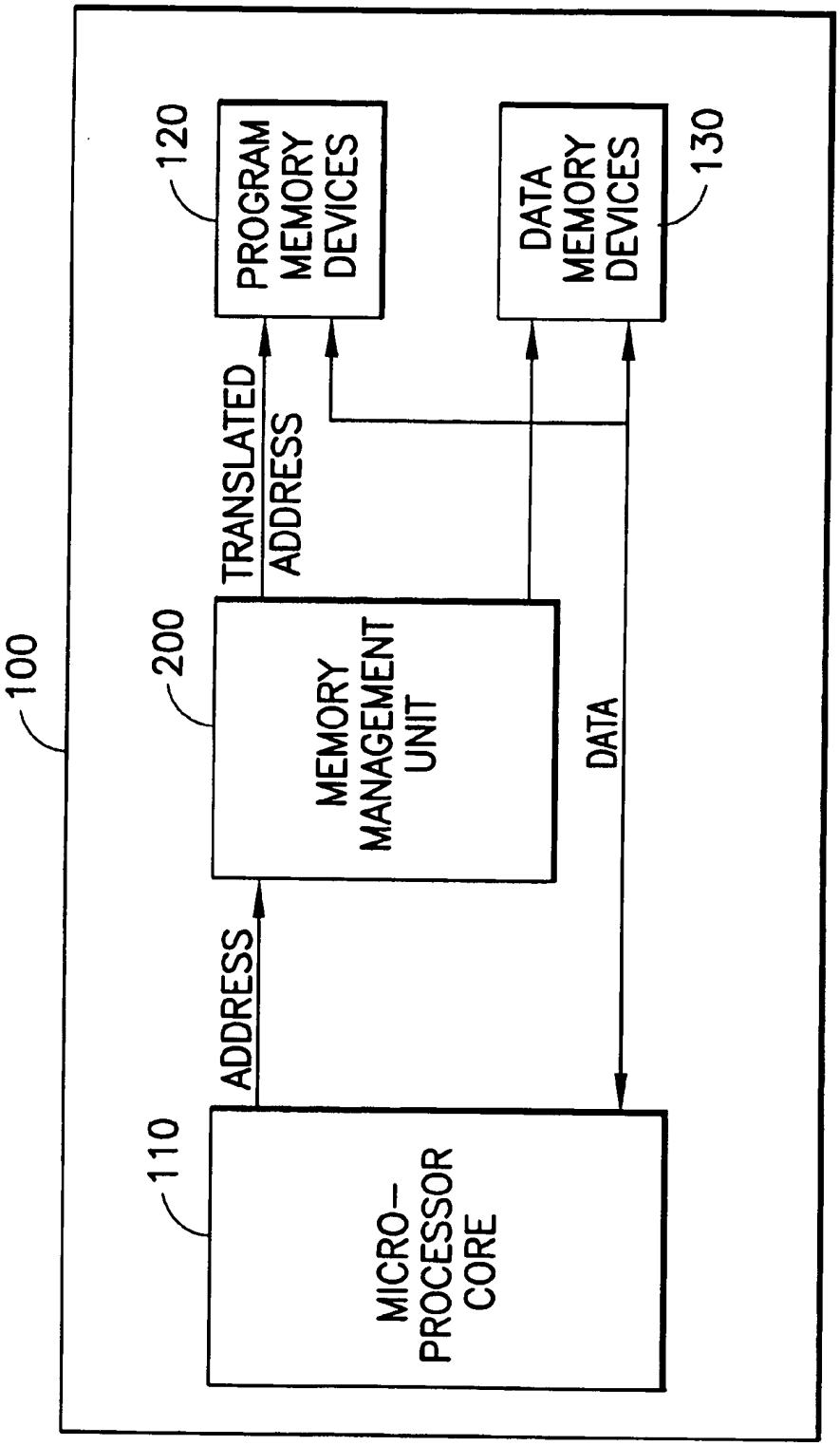

100
120
PROGRAM MEMORY DEVICES
130
DATA MEMORY DEVICES
TRANSLATED ADDRESS
200
MEMORY MANAGEMENT UNIT
DATA
ADDRESS
110
MICRO-PROCESSOR CORE

FIG. 1

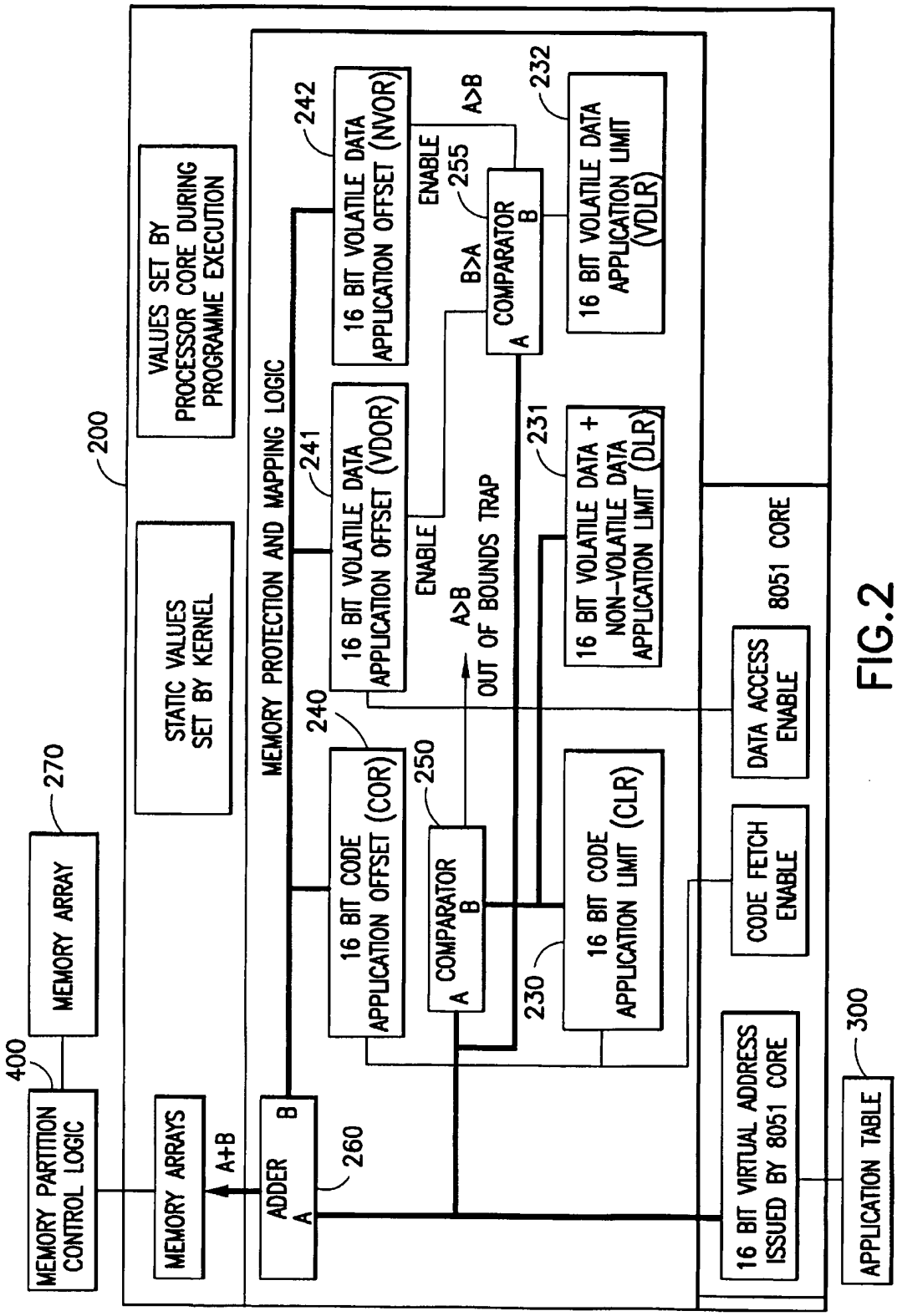

200
VALUES SET BY PROCESSOR CORE DURING PROGRAMME EXECUTION
STATIC VALUES SET BY KERNEL
MEMORY PROTECTION AND MAPPING LOGIC
16 BIT VOLATILE DATA APPLICATION OFFSET (NVOR)
242
ENABLE
A>B
255
B>A
COMPARATOR
A
B
232
16 BIT VOLATILE DATA APPLICATION LIMIT (VDLR)
16 BIT VOLATILE DATA APPLICATION OFFSET (VDOR)
241
ENABLE
A>B
OUT OF BOUNDS TRAP
231
16 BIT VOLATILE DATA + NON-VOLATILE DATA APPLICATION LIMIT (DLR)
240
16 BIT CODE APPLICATION OFFSET (COR)
250
COMPARATOR
A
B
230
16 BIT CODE APPLICATION LIMIT (CLR)
DATA ACCESS ENABLE
8051 CORE
CODE FETCH ENABLE
16 BIT VIRTUAL ADDRESS ISSUED BY 8051 CORE
300
APPLICATION TABLE
ADDER
A
B
260
A+B
MEMORY ARRAYS
MEMORY PARTITION CONTROL LOGIC
400
MEMORY ARRAY
270

FIG.2

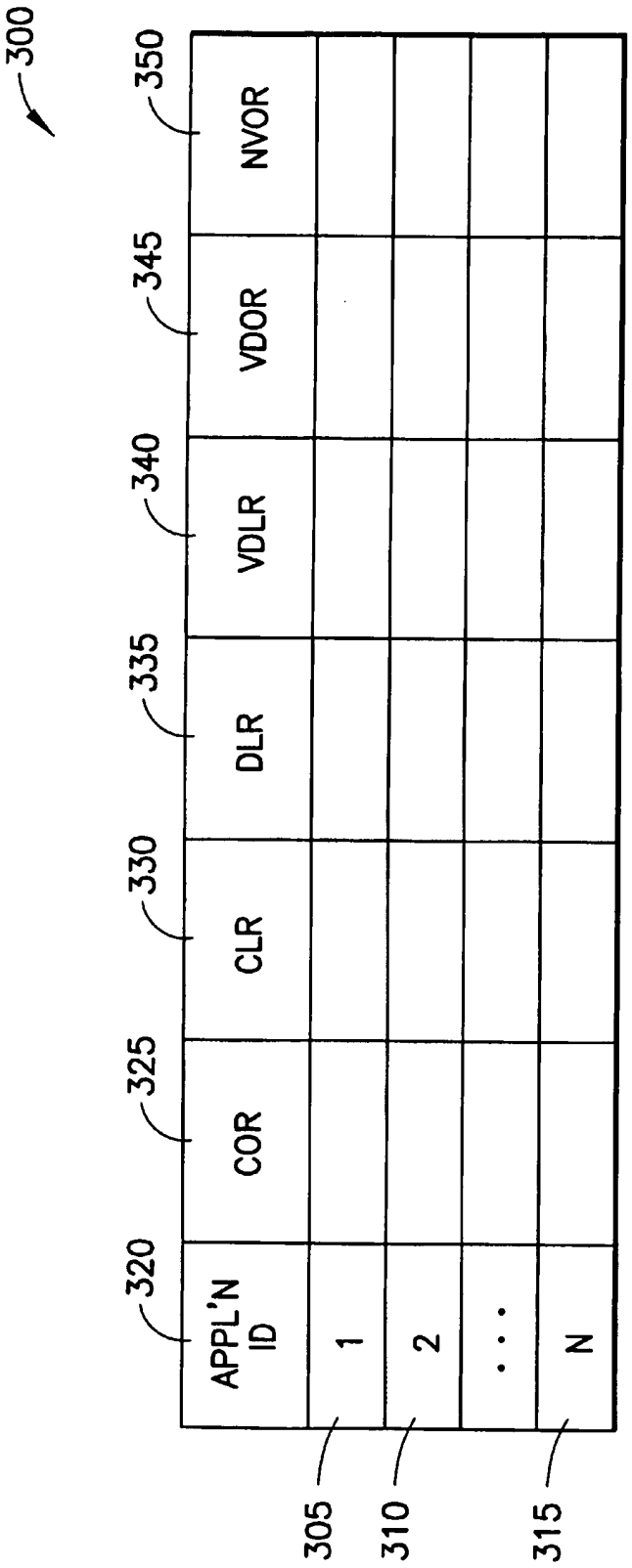

300
320
APPL'N ID
325
COR
330
CLR
335
DLR
340
VDLR
345
VDOR
350
NVOR
1
2
...
N
305
310
315

FIG.3

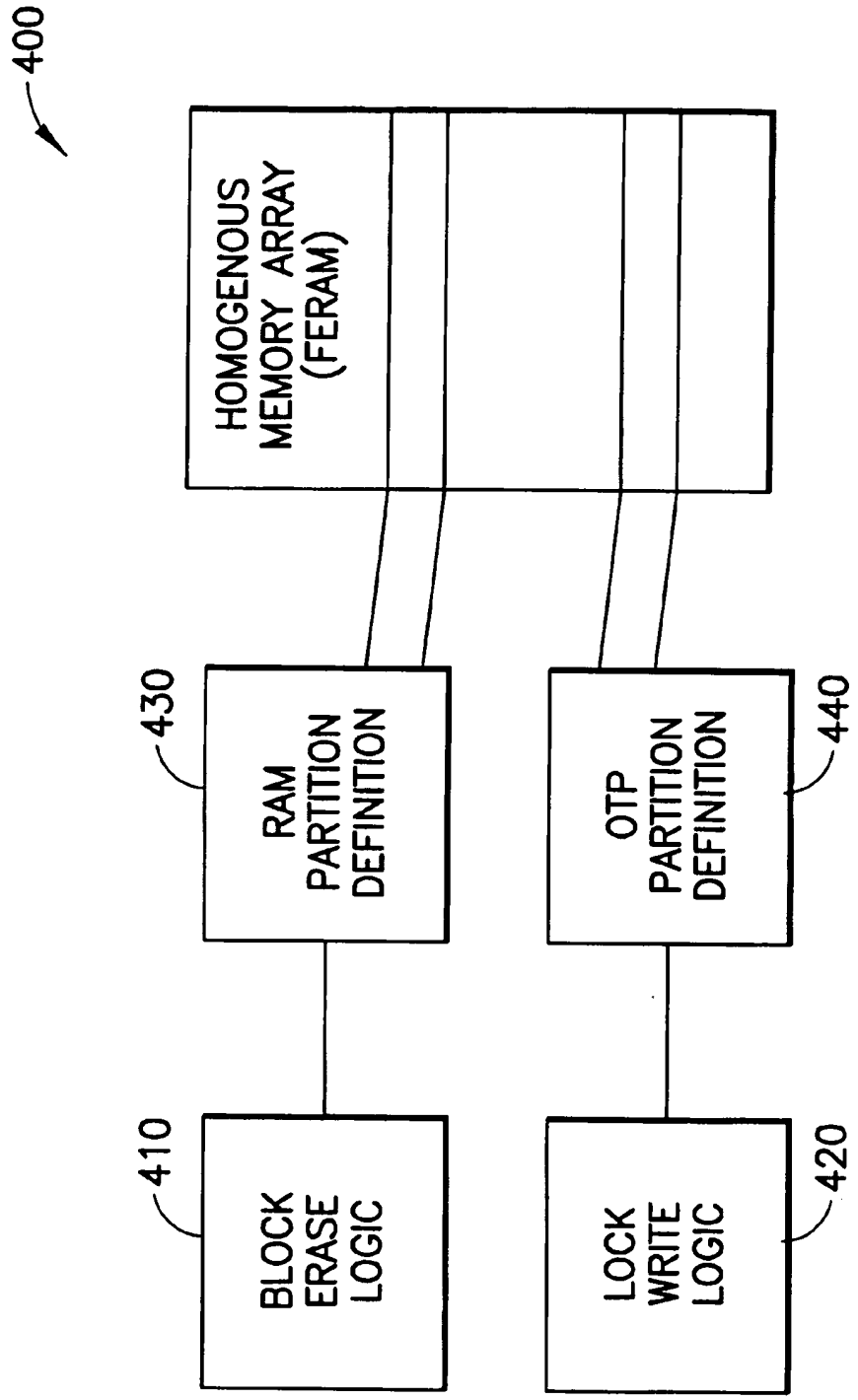

400
HOMOGENOUS MEMORY ARRAY (FERAM)
430
RAM PARTITION DEFINITION
440
OTP PARTITION DEFINITION
410
BLOCK ERASE LOGIC
420
LOCK WRITE LOGIC

FIG. 4

MEMORY MANAGEMENT METHOD AND APPARATUS FOR PARTITIONING HOMOGENEOUS MEMORY AND RESTRICTING ACCESS OF INSTALLED APPLICATIONS TO PREDETERMINED MEMORY RANGES

FIELD OF THE INVENTION

The present invention relates generally to a memory management system for single-chip data processing circuits, such as a smart card, and more particularly, to a memory management method and apparatus that (i) partitions homogeneous memory devices to achieve heterogeneous memory characteristics and (ii) restricts access of installed applications to predetermined memory ranges.

BACKGROUND OF THE INVENTION

Smart cards typically contain a central processing unit (CPU) or a microprocessor to control all processes and transactions associated with the smart card. The microprocessor is used to increase the security of the device, by providing a flexible method to implement complex and variable algorithms that ensure integrity and access to data stored in non volatile memory. To enable this requirement, smart cards contain non-volatile memory, for storing program code and changed data, and volatile memory for the temporary storage of certain information. In conventional smart cards, each memory type has been implemented using different technologies.

Byte erasable EEPROM, for example, is typically used to store non-volatile data, that changes or configures the device in the field, while Masked-Rom and more recently one-time-programmable read-only memory (OTPROM) is typically used to store program code. The data and program code stored in such non-volatile memory will remain in memory, even when the power is removed from the smart card. Volatile memory is normally implemented as random access memory (RAM). The hardware technologies associated with each memory type provide desirable security benefits. For example, the one-time nature of OTPROM prevents authorized program code from being modified or over-written with unauthorized program code. Likewise, the implementation of volatile memory as RAM ensures that the temporarily stored information, such as an encryption key, is cleared after each use.

There is an increasing trend, however, to utilize homogeneous memory devices, such as ferroelectric random access memory (FERAM), in the fabrication of smart cards. FERAM is a nonvolatile memory employing a ferroelectric material to store the information based on the polarization state of the ferroelectric material. Such homogeneous memory devices are desirable since they are non-volatile, while providing the speed of RAM, and the density of ROM while using little energy. The homogeneous nature of such memory devices, however, eliminates the security benefits that were previously provided by the various hardware technologies themselves. Thus, a need exists for the ability to partition such otherwise homogeneous memory devices into volatile, non-volatile and program storage (ROM) regions with the appropriate corresponding memory characteristics.

U.S. Pat. No. 5,890,199 to Downs discloses a system for selectively configuring a homogeneous memory, such as FERAM, as read/write memory, read only memory (ROM) or a combination of the foregoing. Generally, the Downs system allows a single portion of the memory array to be

partitioned as ROM for storing the software code for only an application. In addition, the Downs system does not provide a mechanism for configuring the homogeneous memory to behave like RAM that provides for the temporary storage of information that is cleared after each use.Single-chip microprocessors, such as those used in smart cards, increasingly support multiple functions (applications) and must be able to download an application for immediate execution in support of a given function. Currently, single-chip microprocessors prevent an installed application from improperly corrupting or otherwise accessing the sensitive information stored on the chip using software controls. Software-implemented application access control mechanisms, however, rely on the total integrity of the embedded software, including the software that can be loaded in the field.

Ideally, a system would allow a third party to create an application and load it onto a standard card, which removes the control over the integrity of the software allowing malicious attacks. This may be overcome, for example, by programming an interpreter into the card that indirectly executes a command sequence (as opposed to the microprocessor executing a binary directly). This technique, however, requires more processing power for a given function and additional code on the device which further increases the cost of a cost-sensitive product. A mechanism is required that ensures that every memory transaction made by a loaded application is limited to the memory areas allocated to it. Furthermore, this mechanism needs to function independently of the software such that it cannot be altered by malicious programs. Thus, even malicious software is controlled.

A further need exists for a hardware-implemented access control mechanism that prevents unauthorized applications from accessing stored information, such as sensitive data, and the controlling software of smart cards. Hardware-implementations of an access control mechanism will maximize the security of the single-chip microprocessor, and allow code to be reused, by isolating the code from the actual hardware implementation of the device. Furthermore, a hardware-implemented access control mechanism allows a secure kernel (operating system) to be embedded into the device, having access rights to features of the device that are denied to applications.

SUMMARY OF THE INVENTION

Generally, a memory management unit is disclosed for a single-chip data processing circuit, such as a smart card. The memory management unit (i) partitions a homogeneous memory device to achieve heterogeneous memory characteristics for various regions of the memory device, and (ii) restricts access of installed applications executing in the microprocessor core to predetermined memory ranges. Thus, the memory management unit imposes firewalls between applications and permits hardware checked partitioning of the memory.

The memory management unit provides two operating modes for the processing circuit. In a secure kernel mode, the programmer can access all resources of the device including hardware control. In an application mode, the memory management unit translates the virtual memory address used by the software creator into the physical address allocated to the application by the operating system in a secure kernel mode during installation. The present invention also ensures that an application does not access memory outside of the memory mapped to the application

by the software when in secure kernel mode. Any illegal memory accesses attempted by an application will cause a trap, and in one embodiment, the memory management unit restarts the microprocessor in a secure kernel mode, optionally setting flags to permit a system programmer to implement an appropriate mechanism to deal with the exception.

An application table records the memory demands of each application that is installed on the single-chip data processing circuit, such as the volatile, non-volatile and program storage (OTPROM) memory requirements of each application. The memory management unit implements memory address checking using limit registers and translates virtual addresses to an absolute memory address using offset registers. Once the appropriate memory areas have been allocated to each application program, the memory management unit loads limit and offset registers with the appropriate values from the application table to ensure that the executing application only accesses the designated memory locations.

According to another aspect of the invention, the memory management unit partitions a homogeneous memory device, such as an FERAM memory device, to achieve heterogeneous memory characteristics normally associated with a plurality of memory technologies, such as volatile, non-volatile and program storage (ROM) memory segments. Once partitioned, the memory management unit enforces the appropriate corresponding memory characteristics for each heterogeneous memory type. A memory partition control logic is programmed with the required partitioning associated with each portion of the homogeneous memory in order that the homogeneous memory behaves like volatile, non-volatile and program storage (OTPROM) memory technologies, as desired.

A more complete understanding of the present invention, as well as further features and advantages of the present invention, will be obtained by reference to the following detailed description and drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a schematic block diagram illustrating a single-chip data processing circuit, such as a smart card, that includes a memory management unit in accordance with the present invention;

FIG. **2** is a schematic block diagram of an exemplary hardware-implementation of the memory management unit of FIG. **1**;

FIG. **3** is a sample table from the exemplary application table of FIG. **2**; and

FIG. **4** is a schematic block diagram illustrating the memory partition control logic of FIG. **2**.

DETAILED DESCRIPTION

FIG. **1** illustrates a single-chip data processing circuit **100**, such as a smart card, that includes a microprocessor core **110**, memory devices **120**, **130** and a memory management unit **200** that interfaces between the microprocessor core **110** and the memory devices **120**, **130** for memory access operations. In accordance with the present invention, the memory management unit **200** (i) partitions a homogeneous memory device to achieve heterogeneous memory characteristics for various regions of the memory device, and (ii) restricts access of installed applications executing in the microprocessor core **110** to predetermined memory ranges. It is noted that each of these two features are independent, and may be selectively and separately implemented in the memory management unit **200**, as would be apparent to a person of ordinary skill. In addition, while the present invention is illustrated in a smart card environment, the present invention applies to any single-chip data processing circuit, as would be apparent to a person of ordinary skill in the art.

According to a feature of the present invention, the memory management unit **200**, discussed further below in conjunction with FIG. **2**, imposes firewalls between applications and thereby permits hardware checked partitioning of the memory. Thus, an application has limited access to only a predetermined memory range. As discussed further below, the memory management unit **200** performs memory address checking and translates addresses based on user-specified criteria.

According to another feature of the invention, the memory management unit **200** provides two operating modes for the microprocessor **110**. In a secure kernel mode, the programmer can access all resources of the device including hardware control. In an application mode, the memory management unit **200** translates the virtual memory address used by the software creator into the physical address allocated to the application by the operating system in a secure kernel mode during installation. The present invention also ensures that an application does not access memory outside of the memory mapped to the application by the software when in secure kernel mode. Any illegal memory accesses attempted by an application will cause a trap, and in one embodiment, the memory management unit **200** restarts the microprocessor **10** in a secure kernel mode, optionally setting flags to permit a system programmer to implement an appropriate mechanism to deal with the exception.

In this manner, an exception is identified if an application is written with the accidental or specific intention of compromising the security of the smart card, by accessing stored data, code or by manipulating the hardware to indirectly influence the operation of the chip. The memory management unit **200** limits the application to the allocated program code and data areas. Any other references result in termination of the application and flagging the secure kernel that such an illegal attempt has been made. Thus, each application is isolated from all other applications, the hardware and the secure kernel. In an implementation where application isolation is not needed, the security mechanism acts as a general protection unit trapping software errors.

According to a further feature of the present invention, the memory management unit **200** partitions a homogeneous memory device, such as an FERAM memory device, to achieve heterogeneous memory characteristics normally associated with a plurality of memory technologies, such as volatile, non-volatile and program storage (ROM) memory segments. Once partitioned, the memory management unit **200** enforces the appropriate corresponding memory characteristics for each heterogeneous memory type.

FIG. **2** provides a schematic block diagram of an exemplary hardware-implementation of the memory management unit **200**. As previously indicated, the memory management unit **200** (i) partitions a homogeneous memory device to achieve heterogeneous memory characteristics for various regions of the memory device, and (ii) restricts access of installed applications executing in the microprocessor core **110** to predetermined memory ranges. As shown in FIG. **2** and discussed further below in conjunction with FIG. **4**, the memory management unit **200** includes a section for memory partition control logic **400**. Generally, the memory partition control logic **400** is programmed with the required

partitioning associated with each portion of the homogeneous memory in order that the homogeneous memory behaves like volatile, non-volatile and program storage (OTPROM) memory technologies, as desired. An application would normally be allocated different memory areas for code and data, and the data area can be further divided into a volatile portion, for scratch pad operations, and non-volatile storage areas.

In addition, the memory management unit **200** includes an application table **300**, discussed further below in conjunction with FIG. **3**. Generally, the application table **300** records the memory demands of each application that is installed on the single-chip data processing circuit **100**. For example, the application table **300** indicates the volatile, non-volatile and program storage (OTPROM) memory requirements of each application. The application table **300** is generated by the microprocessor **110** when operating in a secure kernel mode, as each application is installed. The kernel allocates the appropriate memory areas to each application program.

Once the appropriate memory areas have been allocated to each application program, the memory management unit **200** shown in FIG. **2** can load the limit and offset registers **230–232**, **240–242**, discussed below, with the appropriate values from the application table **300** to ensure that the executing application only accesses the designated memory locations. Generally, the memory management unit **200** implements memory address checking using the limit registers **230–232** and translates addresses to an absolute memory address using the offset registers **240–242**.

In addition to restricting access of installed applications executing in the microprocessor core **110** to predetermined memory ranges, the memory management unit **200** also translates addresses between the virtual memory address used by the software programmer into the physical address allocated to the application by the operating system in a secure kernel mode, before it hands over execution to the application code. It is noted that when programming the illustrative 8051 microprocessor, a software programmer starts with a code space starting at an address of 0, and a data space starting at an address of 0. Furthermore, the size of the code and data space is a variable corresponding to the required resource of a given application.

Again, the application has the appropriate volatile, non-volatile and program storage (OTPROM) memory allocations that are translated and checked by the memory management unit **200**, in a manner described below, such that attempts to access memory outside the designated memory area will result in the application being terminated. The kernel will be restarted and the offending trapped access, being stored for interrogation by the kernel.

The hardware memory-mapping scheme and out of area protection hardware mechanism is shown in FIG. **2**. In the illustrative 8051 microprocessor, only one application is active at any time, so only one set of mapping logic is required, as shown in FIG. **2**. Thus, the microprocessor core **110** must implement context switching in a multi-function environment, as would be apparent to a person of ordinary skill. As previously indicated, the memory management unit **200** includes a pair of limit and offset registers, such as the registers **230–232**, **240–242**, respectively, for each memory technology that is managed by the memory management unit **200**.

Before an application is started, the associated memory requirements are retrieved from the application table **300** by the secure operating system running in the kernel mode. The associated memory requirements are loaded into the corresponding limit and offset registers **230–232**, **240–242**.

Thereafter, the kernel loads the code application offset register (COR) **240** with the address of where the application program code is stored in memory. The kernel then loads the code application limit register (CLR) **230** with the size of the application code space. Similarly, the data space can be defined as a block of memory, whose size is the sum of the sizes of both the volatile and non-volatile memory, allocated to that application. Thus, the kernel loads the data limit register (DLR) **231** with the size of the data space (both the volatile and nonvolatile memory). The size of the allocated volatile memory is loaded into the volatile data limit register (VDLR) **232**, and the base address to be used for the scratch pad memory (RAM) is loaded into the volatile data offset register (VDOR) **241**. Finally, the base address to be used for non-volatile storage (EEPROM) allocated to the application is loaded into the non volatile offset register (NVOR) **242**.

In one implementation, the memory protection mechanism checks the virtual memory addresses assigned by the programmer, as opposed to the absolute addresses allocated by the kernel. Thus, the illegal access mechanism is simplified, as an illegal memory access is identified when an access is made to a location having a virtual address that is greater than the value contained in the appropriate limit register. Thus, as shown in FIG. **2**, the memory management unit **200** contains comparators **250**, **255** for comparing the virtual address issued by the microprocessor core **210**, to the value contained in the appropriate limit register **230–232**. If the application is attempting an unauthorized memory access, the corresponding comparator **250**, **255** will set an out-of-bounds trap.

If the application is attempting an authorized memory access, the corresponding comparator **250**, **255** will enable the appropriate offset register **240–242**, and the value from the offset register will be added by an adder **260** to the virtual address issued by the microprocessor core **210**. In one preferred implementation, the limit and offset registers **230–232**, **240–242** and the comparators **250**, **255** are fabricated using known tamper-resistant technologies to preclude physical security attack.

FIG. **3** illustrates an exemplary application table **300** that stores information on each application installed on the single-chip data processing circuit **100**, including the memory demands of each installed application. As shown in FIG. **3**, the application table **300** indicates the volatile, non-volatile and program storage (OTPROM) memory requirements of each application. The application table **300** may be generated by the microprocessor **110** when operating in a secure kernel mode, as each application is installed. The kernel allocates the appropriate memory areas to each application program.

The application table **300** maintains a plurality of records, such as records **305–315**, each associated with a different application. For each application identifier in field **320**, the application table **300** includes the base address of where the application program code is stored in memory, and the corresponding size of the application code space in fields **325** and **330**, respectively. In addition, the application table **300** indicates the total size of the data space in field **335** (sum of both the volatile and non-volatile memory), with the size of the allocated volatile memory stored in field **340**, the base address for the scratch pad memory (RAM) in field **345**, and the base address for non-volatile storage (EEPROM) is recorded in field **350**. As previously indicated, when an application becomes active, each of the corresponding memory range values from fields **325** through **350** are retrieved and loaded into the appropriate limit and offset registers **230–232**, **240–242**, respectively.

FIG. **4** illustrates the memory partition control logic **400** for a homogeneous memory array **450**. As previously indicated, the memory partition control logic **400** contains registers associated with each portion of the homogeneous memory in order that the homogeneous memory behaves

like volatile, non-volatile and program storage (OTPROM) memory technologies, as desired. An application would normally be allocated different memory areas for code and data, and the data area can be further divided into a volatile portion, for scratchpad operations, and non-volatile storage areas. FERAM is inherently a non-volatile array. In other words, FERAM can be changed many times and holds the last written value, even when powered down, in a manner similar to EEPROM. Thus, it is unnecessary to force EEPROM-behavior onto the FERAM to achieve a non-volatile array.

To create a volatile array using the non-volatile FERAM array, erase circuitry **410**, **430** is added, for example, by writing 0's to each address, or using a block erase feature built into the array that writes 0's to many addresses in parallel. The erase circuitry **410**, **430** records the upper and lower limits of the memory range that should behave like a volatile array. Similarly, to ensure that the code is not written to, a write inhibit has to be forced onto the memory array using lock-write circuitry **420**, **440**. The lock-write circuitry **420**, **440** records the upper and lower limits of the memory range that should behave like program storage (OTPROM) memory.

Once the application space has been setup by the secure kernel, defined areas of the homogeneous array need to behave in the appropriate manner. This can be achieved by mapping the erase logic using the same memory definitions used to define the volatile memory area for applications. Before an application is started (or after or both), the erase mechanism is enabled, ensuring that an application when started can see no residual values left over by a previous application or the kernel, that may have used the designated block. Similarly, the same simple mechanism can be used to enforce a write-lock on the area designated as the code space for the application to prevent the application from modifying its code to cause potential unknown conditions and hence revealing secure aspects of the device.

The application RAM area is defined by parameters loaded into erase circuitry **430**. Typically, the value loaded into the erase circuitry **430** would be the physical address location within the FERAM memory array and the size of the allocated memory. The block erase logic **410**, when activated, is constrained by the erase circuitry **430** to erase the predefined area. The same principle is used to obtain OTP characteristics. OTP partitioning is defined by the lock-write circuitry **440**, which allocates an area of the same memory array once parameters are loaded. The lock write logic **420** removes the write capability for the area defined in the lock-write circuitry **440** giving the area the same characteristics as OTP memory.

It is to be understood that the embodiments and variations shown and described herein are merely illustrative of the principles of this invention and that various modifications may be implemented by those skilled in the art without departing from the scope and spirit of the invention.

I claim:

1. a single-chip data processing circuit, comprising:

a processor for executing a plurality of applications;

a homogeneous memory device; and

a memory management unit for (i) partitioning said homogeneous memory device for each of said plurality of applications to achieve memory characteristics associated with a plurality of memory technologies, including a volatile memory technology, (ii) recording for each of said applications a range for an assigned heterogeneous memory type corresponding to each of said partitions, and (iii) enforcing memory characteristics for a heterogeneous memory type corresponding to each of said partitions for each of said applications.

2. The single-chip data processing circuit of claim **1**, wherein said memory technologies include a read only memory technology with limited programmability.

3. The single-chip data processing circuit of claim **1**, wherein said memory technologies include a non-volatile memory technology.

4. The single-chip data processing circuit of claim **1**, wherein said memory management unit includes block erase logic to achieve volatile memory characteristics.

5. The single-chip data processing circuit of claim **1**, wherein said memory management unit includes lock-write erase logic to achieve memory characteristics with limited programmability.

6. The single-chip data processing circuit of claim **1**, wherein said memory management unit further comprises a register for storing a base memory address corresponding to a location where a non-volatile memory region begins.

7. The single-chip data processing circuit of claim **1**, wherein said memory management unit further comprises a register for storing a memory address corresponding to a location where a non-volatile memory region ends.

8. The single-chip data processing circuit of claim **1**, wherein said memory management unit further comprises a register for storing a base memory address corresponding to a location where a volatile memory region begins.

9. The single-chip data processing circuit of claim **1**, wherein said memory management unit further comprises a register for storing a memory address corresponding to a location where a volatile memory region ends.

10. A method for partitioning a homogeneous memory device to achieve heterogeneous memory characteristics for various regions of the memory device for a plurality of applications, comprising the steps of:

partitioning said homogeneous memory device to achieve memory characteristics associated with a plurality of memory technologies, including a volatile memory technology;

recording for each of said applications a range for an assigned heterogeneous memory type corresponding to each of said partitions; and

enforcing memory characteristics for a heterogeneous memory type corresponding to each of said partitions for each of said applications.

11. The method of claim **10**, wherein said memory technologies include a read only memory technology with limited programmability.

12. The method of claim **10**, wherein said memory technologies include a non-volatile memory technology.

13. The method of claim **10**, further comprising the step of erasing a partition of said homogeneous memory device to achieve volatile memory characteristics.

14. The method of claim **10**, further comprising the step of preventing write operations in a partition to achieve memory characteristics with limited programmability.

15. The method of claim **10**, further comprising the step of storing a base memory address corresponding to a location where a non-volatile memory region begins.

16. The method of claim **10**, further comprising the step of storing a memory address corresponding to a location where a non-volatile memory region ends.

17. The method of claim **10**, further comprising the step of storing a base memory address corresponding to a location where a volatile memory region begins.

18. The method of claim **10**, further comprising the step of storing a memory address corresponding to a location where a volatile memory region ends.

* * * * *